EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH
CERN — SL DIVISION

CERN SL-99-016 OP

The Xdataviewer Programmer's and User's Guide

Morpurgo, G

Abstract

This report describes in detail all the features of the Xdataviewer, a software graphical package developed in the CERN/SL Division and widely used in the LEP and SPS Control Rooms.Goal of the Xdataviewer is to help the Application Programmers to present numerical data both in graphical or in alphanumerical format, and Users to look at these data and to possibly interact with them. Sophisticated built-in Zoom and Data Editing capabilities are implemented, as well as a flexible Data Output generation facility. A complete C Callable Interface is provided, including routines to inform the Application Program about the User's interaction with the displayed data. The data to be displayed, formatted using the MOPS[1] formalism, can be contained in a Unix shared memory, or in a file, or can be found on the Web. The package has been written in C on a Unix HP_UX platform, making use of the standard X Window libraries (Xlib, Xt, Motif). It can be either run as a stand alone process, or it can be embedded in the Application Program itself.

Presented at PAC-99, 1999 Particle Accelerator Conference
New York
29 March-2 April 1999

Geneva, Switzerland
March, 1999

Contents

Preface

The CERN/SL Xdataviewer is a powerful tool to display, edit and interact with data arrays. Its original design goes back to the late 80's, when the first APOLLO workstations were introduced in the SPS Division, and a graphical tool (the Dataviewer) had to be implemented, to reproduce and extend the functionality of facilities existing in the SPS Nodal environment. The first APOLLO workstations were runnning a proprietary operating system, with embedded Unix-like facility. They were supporting machine-dependent graphics primitives (no standard was available, and the notion of portability was not really a fashionable one). As everyone can imagine, those workstations were impressive at that time, but slow and small for today's standards. Many applications were anyway built using the Dataviewer, which played a central role in the daily operation of the SPS and then of the LEP. Main customers of the Dataviewer were the Function Editor program, and the Orbit Display and Correction package, whose requirements strongly influenced the Dataviewer development.

Many slightly different variants of the Dataviewer were developed by different people, until 1990, when Ann Sweeney[1] cleaned and enhanced the code, merging the best features of this "parallel development", and froze the product. The Dataviewer represented, for many application programmers who wanted to be focused on solving their problems rather than on reinventing graphics packages, a way of saving many man-years.

Around 1994 the APOLLO workstations were replaced by HP-UX and Xterminals, running UNIX® and X Window™. Suddenly the Dataviewer became unusable, together with a huge amount of application programs based on it. As I was responsible for many of these programs, I decided to port the Dataviewer (~15k lines of C-code and APOLLO specific calls) to the new X-Window environment. This operation was less difficult than expected, also due to the clean way in which the Dataviewer software had been structured by Ann. I called the new version "Xdataviewer". As soon as the Xdataviewer was made available, more and more applications used it, and many new features and facilities were added to the tool, as a consequence of new wishes expressed by the users and by the application programmers. This development was always done maintaining the backwards compatibility, so that already existing applications could use the latest versions without any modification to the source code. Also, the documentation for these new features was always kept up to date [2].

Some of the new features clearly extend a lot the range of action of the Xdataviewer. For instance, the "Embedded Dataviewer", where the Xdataviewer, instead of being run as a stand alone process, is fully contained within the Application Program itself. Or, more recently, a new drawing facility, developed to add pictures and text to the data displayed. Or else, the possibility of loading the data from files, and of navigating through a sequence of files. All these enhancements more than doubled the lines of C-code which constitute the Xdataviewer and its libraries.

All of these novelties fully justify, to my opinion, a complete rewriting of the whole documentation, to present the Users and the Application Programmers with a complete overview of this products.

UNIX is a registered trademark of AT&T
The X Window System is a trademark of the Massachusetts Institute of Technology

1

Introduction

The Xdataviewer is a tool providing a large set of facilities to help the Application Programmer to present data in graphical format, and the User to zoom, edit, list, plot, print and save the data. The Xdataviewer can be run either as a standalone process (controllable from an application process via Unix signals), or embedded as a part of the Application Program itself.

The Xdataviewer requires the data to be set up, in the framework of the MOPS formalism [3], with the help of the functions defined in the Callable Interface described later in this manual. The data can be found either in a real MOPS shared memory area, or in a file where a MOPS area was previously saved in binary format, or in an internal memory buffer structured as a MOPS (Embedded Dataviewer only). As long as the required data format is respected, the Xdataviewer can also access data on the World Wide Web..

Chapter 2 passes in review the different concepts extensively used later, and explains the way in which the Xdataviewer works. Properties of different objects are listed, and the routines used to modify these properties are mentioned. These routines will be described in greater detail in chapter 3, where we examine the Xdataviewer Callable Interface.

Chapter 4 describes the command line arguments for the Xdataviewer, and the different problems which can be encountered when the Xdataviewer has to be started together or from an Application Program.

The User can interact with the data in many different ways, using the many possibilities offered by the Xdataviewer Motif Graphical User Interface, described in chapter 5.

Chapter 6 is dedicated to explain how to insert the Embedded Dataviewer inside an application program. Emphasis is given to the differences between the standalone version and the embedded one.

The different ways in which the Xdataviewer can interact with the Application Program will be treated in detail in chapter 7.

Chapter 8 contains some useful hints and tricks for Application Programmers, while chapter 9 contains many simple example programs, dealing with the different problems which the Application Programmer can encounter.

The picture drawing facility will be treated extensively in chapter 10, as it constitutes a useful development very well decoupled from the rest.

Chapter 11 describes the Help facility, via which an Application Programmer can provide on-line help to the application's Users.

This User guide is completed by an Appendix, where some constants used by the Xdataviewer are listed, and a short reference to the MOPS library is provided.

2

A General Overview

2.1 Concepts

The Xdataviewer understands four basic entities - **views**, **graphs**, **plots** and **objects**. These are described below.

An **object** is a single array of data. It can be an array of integer or floating point numbers, or of text strings. It has a *length* (i.e. number of elements) and the *data*. Every element of the array is of the same type. In the Xdataviewer implementation, an object is stored and handled as a MOPS object. This simplifies the task of the Xdataviewer, because the MOPS library will take care of things like keeping the length of the object, allocating new space if elements have to be added to the object, etc.
As we will see later, some objects represent numerical data to be displayed, while others contain auxiliary information (colours, labels, error bars, etc.) . The Application Program is responsible for filling the objects with useful data.

A **plot** is a set of points, a line or an histogram constructed from one or two **objects**. A plot from one object consists of the value of each element of the array (y-axis) plotted against the element number (x-axis). A plot from two objects consists of the value of each element of the first array (x-axis) plotted against the value of the second array (y-axis).
Besides its data object(s), a plot may be better adapted to the needs of the application by setting its **properties** (colours, labels, etc.) with the help of functions in the Callable Interface, and of *auxiliary* objects. We will describe more in details the plot properties in section "*The plot and its properties*".
In order to be displayed, a plot needs to be *attached* to a **graph** currently displayed. A plot can be attached to any number of graphs (there is, however, a limit to the number of plots which can be attached to the same graph).

A **graph** represents a window on the *x,y* space, in which one or more **plots** can be shown. The limits in x and y defining the visible portion of the x,y space can be imposed by the program, or else automatically adapted to entirely accommodate all the data points of the plots attached to the graph. The User can also change these limits, by using the zoom facility. When a graph is displayed, all the points of the attached plots which fit in the visible window will be shown. In section "*The graph and its properties*" we will describe more in detail the characteristics of a graph.
In order to be displayed, a graph needs to be *attached* to the **view** currently displayed. A graph can be attached to any number of views (there is, however, a limit to the number of graphs which can be attached to the same view).

A **view** is a collection of one or more **graphs**. In practice, it defines what can be shown in the graphical part of the Xdataviewer at a given time. It is possible to define more than one view, but only one can be displayed on the screen at a time. The view to be selected can be chosen either by the User (through the Xdataviewer interface), or by the application program (through one of the functions in the Callable Interface). When a view is selected, all the graphs attached to it are displayed (the Xdataviewer screen is subdivided to accommodate all of them).
Most of the Xdataviewer routines are therefore concerned with creating plots, graphs, views, tailoring them accordingly to the application's needs, and attaching plots to graphs and graphs to views.

2.2 All what you need to know about MOPS.

A MOPS is a memory buffer containing data objects accessible by names. A library of C routines (the "MOPS library") makes the creation of the MOPS and the access to it, as well as the creation, access and manipulation of the data objects, straightforward operations. A MOPS can be contained in a Unix shared memory, or inside a memory buffer internal to a process. It can be saved as it is into a file, and reloaded from it.
Basic concepts in the MOPS are :

- The **"dummy" file**, needed for Unix Shared Memory Identification. In order to be able to unambiguously identify a piece of shared memory via a name, the Unix system requires the presence of a dummy file (zero length is OK). The MOPS routines requiring the name of the dummy file as an argument follow this convention : if the argument starts by the @ character, that character will be skipped and the rest will be used as a complete path name to the dummy file. Otherwise the argument will be used as the dummy file name, to be looked for in directory $HOME/shardat.
- The **MOPS pointer**. This is a (char *) variable (called **q**, by tradition), either returned by the creation of a new MOPS (routine *c_sdalloc*), or by the access to an already existing MOPS (routine *c_sdacc*), or representing the pointer to an internal memory buffer (either static, or allocated with *malloc*). This pointer will be an argument in most of the MOPS or Xdataviewer functions.
- The **MOPS size**. Specified at the creation, it must be large enough to contain all the data objects, plus their directory entries. If the MOPS has to contain Xdataviewer objects (views, plots, graph), some space will have to be foreseen for them.
- The **initialization of the MOPS directory**. Routine *c_sdini* enables the Application Programmer to define into how many objects the MOPS will be structured. If the MOPS has to be used with the Xdataviewer, the number of object should also include 5 entries for the Xdataviewer objects (views, graphs, plots, help, communication)
- The **creation of the MOPS objects**. This is done by routines *c_sdbook* or *c_sdzero*.
- The **access to the MOPS objects**. Routine *c_sdptr* will return a pointer to the data corresponding to the specified object. This pointer, casted in the proper way, can be used as a normal array pointer.
- How to **leave** (*c_sdquit*) or to **kill** (*c_sdkill*) **a MOPS**. These are only meaningful for the shared memory instantiation of the MOPS. Notice that after a MOPS is left, the pointer *q* cannot be used anymore, before the MOPS is accessed again. Some of the Xdataviewer routines (e.g *dv_kick*) leave the MOPS, and require it to be accessed again before further operations with it are performed.

This is just the minimal subset of the MOPS library that a Xdataviewer programmer needs to know. Reference[3] contains the complete MOPS documentation.

2.3 The different input sources for the Xdataviewer.

To perform its task of displaying information, the Xdataviewer must have access to the definitions of views, graphs, plots, and to the data of the objects associated with the plots. We refer to the place containing this information as the "**data area space**".
Originally, the Dataviewer could only display data which was contained in a real Unix Shared Memory MOPS. This was mainly due to two facts : *a*) the Dataviewer existed only in the standalone version, and it needed some way to share the data space with the Application Program, and *b*) the performance of the first workstations was not good enough to accept a data communication based on a frequent rewriting of large binary files. Those limitations are now overcome, and in the current Xdataviewer release two new input sources (to be used as *data area space*) have been implemented : the *internal memory buffer*, which can be used to establish the communication between an Application and its Embedded Dataviewer part, and the *MOPS saved into a file* (both for the embedded and for the standalone implementation). It is also possible to switch between the MOPS and the file mode on line, as well as defining a *sequence of files* which can be loaded in sequential or random order. Finally, it is possible to start the Xdataviewer without specifying any data attachment ("*empty*" mode), and only later load from the desired source. A more detailed description of these different possibilities will be found in chapters "How to start the Xdataviewer" and "The Xdataviewer Graphical User Interface".
Another additional flexibility is provided by the possibility of storing the data of the different objects associated with a plot (see above) into independent files (instead then in the *data area space*). The Xdataviewer will recognize that this option has been selected by looking at the first character of the object name (= '*'), and will use the rest of the object name as a path to the file containing the data.
A natural extension of the "loading from files" mechanism is the possibility of accessing *objects*, or entire *data area spaces*, over the Web. When the name of a file to be used by the Xdataviewer starts by two '/' characters, the Xdataviewer will consider it as an "URL", and will look for it over the Web.
Finally, it is possible to replace the currently displayed *data area space* with a new one directly from the Application Program, using the routine *dv_newdata_set* .

2.4 The view and its properties.

A view is always identified by a *viewname*, used as argument in any function referring to that view.
The basic function of a view is that of defining the layout for a graphical page. In fact, when creating a view (*dv_vwcreate*) the programmer is able to define the number of *rows* and *columns* in which the graphical page will be subdivided. This subdivision can be used to specify the position and the size of a graph on the screen, when the graph is attached to the view (*dv_grattach*).
Other two properties of a view are the *menuentry* (a text string identifying the view in the views list) and the *viewtitle* (a text string displayed on the left top corner of the graphical page when the view is selected). Both of them are defined when the view is created, and can be modified at any time (*dv_vwchnames, dv_vwchtitle*).
In order to be accessible to the User, a view's menuentry must be present in the views list. This is automatically done when a view is created. Sometimes, however, it is convenient to temporarily remove one or more views from this list. This operation, and the further re-insertion in the list, can be done by *dv_vwhide* and *dv_vwshow.*

A dynamic property which applies to the currently selected view is the notion of "*view changed*". For instance, if one of the plots currently shown on the screen has been edited, the *view has changed*, and it is necessary to ask the User if he wants to save it before selecting another View. Using the routine *dv_status_set* it is also possible to make the Xdataviewer thinking that the view has changed, even if this is not the case. The same routine can also be used to impose the automatic saving of any changed view, without asking for the User's confirmation.

2.5 The graph and its properties.

A graph is always identified by a *graphname*, used as argument in any function referring to that graph.
The basic function of a graph is that of defining a visible window on the x,y plane, where one or more data arrays (plots) can be displayed. A title, and labels for the x and y axis, are provided when creating the graph (*dv_grcreate*) and can be modified at any time (*dv_grchnames*). It is also possible to specify up to 5 text strings, to be displayed on the left top part of the graph (*dv_grlabels*).
The visible window on the x,y plane may have imposed boundaries in x and y (*dv_grlimits*), or it may automatically rescale to accommodate the range of values of the data in the plots attached to the graph. In this latest case, a margin is left around the data, that can be modified by the routine *dv_grpercent*. These limits will be modified if the zoom facility is used.
The scaling for the x and y axis can be independently set to linear or logarithmic (*dv_grscales*). The x axis may also use time as scaling factor.
A self-adaptive numeric grid can be added to the graph (*dv_grsetgrid*), as well as x and y axis (*dv_grsetzeroline*) . These features can also be interactively set by the User, acting on the Xdataviewer Graphical User Interface.
A special property needed only by particular applications is the maximum number of points for a graph (*dv_grnpoints*). If any plot attached to the graph has more points than the limit *npoints* specified for the graph, only the first npoints will be displayed.
The plots attached to a graph can be displayed in graphical format, or in text format (*dv_grlist*, and *dv_grlistmerge* for graphs containing plots with different number of points).
As mentioned before, a graph can be attached to one or more views (*dv_grattach*), and position and size of the graph on the graphical page can be specified. These attachments can be dynamically removed (*dv_grdetach*) and recreated at any time.

2.6 The plot and its properties.

A plot is always identified by a *plotname*, used as argument in any function referring to that plot.
A plot defines a sequence of x,y points, which have to be plotted in a way depending on the plot properties. The data defining the x and y coordinates are contained in *data objects*, whose names are specified in the plot creation routine (*dv_plcreate2*, or *dv_plcreate1* if one wants the index of the y-data array to be used as x-coordinates).
The first property of a plot is its type, also defined at creation, which can be one of the following :

- DV_MARKER : every point will be displayed as a marker.

- DV_LINE : every point will be displayed as marker. Consecutive points will be joined by straight lines.
- DV_HISTO : every point will be displayed as an histogram bar.
- DV_MULTIMARKER : equivalent to DV_MARKER, but it is possible to specify a different marker for each point.
- DV_MULTIMARKERLINE : equivalent to DV_LINE, but it is possible to specify a different marker for each point.

These two latter types have been introduced especially for monochromatic output (for instance, written proceedings of conferences).
Another type, DV_PICTURE, is available, but plots of this kind are treated in a completely different way, fully described in chapter "*The picture drawing facility for the Xdataviewer*".
By means of the routines *dv_plhis*, *dv_pllin*, *dv_plmar* it is possible to change the colour and the marker type of all the points of a plot. A more flexible possibility is offered by routine *dv_plcol*, in which an *auxiliary object* is defined. The elements of the auxiliary object will be interpreted as colour codes, to be associated with corresponding elements of the data objects. In the case of a plot of type DV_LINE, the lines joining data points will also be coloured accordingly to the elements of the auxiliary object. To give all the lines the same colour, and still an independent colour to the data point markers, use also *dv_plcolouredline*.
Another auxiliary object (of type text) is used in routine *dv_pltxt*, via which a label can be associated with every point in the plot. This labels will be displayed on the bottom part of the graph, together with the data points and the cursor position.
Another property of the plot is the *dataformat* (*dv_pldtform*), defining the shape of the sequence of the x-coordinates of a plot. The dataformat for a plot can have one of the following values :

- DV_RANDOM (no criteria)
- DV_XMONOTONIC (x-coordinates ordered in an increasing order)
- DV_XUNIFORM (to be used for histogram plots with uniformly spaced x values , created with *dv_plcreate2*)

The choice of one or another of the dataformats influences the strategy with which the Xdataviewer searches for the data point closest to the cursor. The processing will be faster for DV_XMONOTONIC plots, so care should be paid in declaring as such all large plots where the x-coordinates are monotonically increasing.
Four other auxiliary objects can also be used to define errorbars to be associated with every datapoint (*dv_plerrorbars*).
A plot has a number of points, which is defined by the size of the data objects specified at its creation. Not used often, routine *dv_plnpoints* enables the Application Programmer to define as visible just a subset of the plot.

Other important properties of the plot concern its representation in text format, which can be imposed by default using routine *dv_pllist*. If a plot is represented in this way, an independent colour can be used for each element by using an auxiliary object (routine *dv_plcoltxt*). The format to be used for printing the data values can also be specified (routines *dv_plprecision* or *dv_pllistform*).

Last but not least, an important property of the plot is its *editability* (routine *dv_plreadwrite*). Only if this property is set, the plot can be edited by the User. It is also possible to make a plot editable and to impose limits on some editing operations.

2.7 Overriding the default properties for graphs and plots.

At their creation, graphs and plots are assigned a default set of property values. These values can be modified, plot by plot and graph by graph, by means of the routines mentioned in the two previous sections. This is impractical if many plots or graphs should be assigned the same property values. Routines *dv_set_default* , *dv_set_defaultlimit* and *dv_set_defaultformat* can be used to modify these default values. The new defaults will be used for all the plots or graphs created after the modification.

2.8 How does the Xdataviewer work : the internal data representation of views,

graphs, plots, objects.

To better understand the behaviour of the Xdataviewer, it is worth mentioning the following thing : most of the time the Xdataviewer is not directly accessing the data buffer (shared memory or internal memory buffer). This access is only needed in the following cases : a) the Xdataviewer starts, b) a new View has to be displayed, c) the Application informs the Xdataviewer that new data are available for display, and d) the contents of a View have to be saved (e.g. after editing operations). In cases **a**), **b**) and **c**) the Xdataviewer accesses the data buffer to **get** information : it finds the View to be displayed, copies into dynamically allocated internal space all the information defining the graphs and plots concerned, allocates space to contain all the data objects and auxiliary objects involved in the plot definitions, and copies all the object elements inside this space (transforming all numerical values into double precision numbers). Then it stops accessing the data space, and it displays the View using the copy made in its internal space. Once the View is not needed anymore, all the dynamically allocated areas are released, to avoid uncontrolled process size growth. In case **d**), the Xdataviewer **puts** information into the data space, by updating there the objects which have been internally modified by editing operations.
This explains why editing operations performed on the displayed data do not affect immediately the data space, and why modifications to the data contained in the data space are not immediately visualized on the screen.

2.9 The different ways of updating the data shown on the screen.

As mentioned above, when the Application declares that new data is available, all the information needed to rebuild the current view needs to be reread from the data space. There are, however, four different strategies which can be used by the Xdataviewer (use *dv_set_update_mode* to select one of these strategies) :

- DV_UPDATE_VIEW : in this case, the view will be redisplayed from scratch, with its initial zooms and subviews.
- DV_UPDATE_DATA : zooms and subviews currently set by the User will be preserved, and only the data points will be redrawn.
- DV_UPDATE_POINTS : in this case, also the existing data points will be left unchanged, and eventual new points will be added to the plot. This could save time, especially in case of large plots.
- DV_UPDATE_ONE_POINT : only one new point will be added. This mode and the previous one should not be used if graphs using the automatic scaling are involved.

2.10 Embedding the Xdataviewer inside the Application Program

Originally, the Xdataviewer existed only as a standalone process. Since version 4, however, the code needed to implement the graphical and control part of the Xdataviewer has been extracted and gathered in a library of routines. Calling one of these routines, an X-Window Application is able to create all the widgets which constitute the Xdataviewer functionality inside its own interface. The main differences between the standalone and the embedded versions derive by the fact that this latest is contained in the same process as the Application, simplifying some interaction and offering, for instance, the option of having the data area contained in a memory buffer directly allocated in the process space. The Embedded Dataviewer will be described in more detail in a dedicated chapter.

2.11 The Xdataviewer and the Application : an introduction to their interaction.

There are several ways in which the Xdataviewer and an Application can interact. In the simplest case an Application can create a shared memory area, and then the Xdataviewer can be run to display its contents.
The interaction between the Xdataviewer and the Application really begins in the slightly more complex case in which the Application wants to update the data contained in the MOPS, and have the Xdataviewer always displaying the most recent data. Another usage is where the User of the Xdataviewer edits or selects the data, and then he wants to notify the Application about its actions, because the Application might need to act upon this changed information. All these cases of exchange of information are implemented using the Unix signal SIGUSR1 (from the Application to the Xdataviewer or viceversa). An

exception is foreseen for the Application -> Xdataviewer interaction for the Embedded Dataviewer. In this case, when the Application wants the Dataviewer part to refresh the data displayed, being part of the same process it calls directly the Dataviewer routine which does that.
The subject of the interactions between the Xdataviewer and the Application will be addressed more in detail in chapter 7.

3

The Xdataviewer Callable Interface

In this chapter are listed and explained all the Xdataviewer functions available to the Application Programmer. The type of the arguments to the different functions will always be explicited, with the following exception, defined here :

- (char *)**q** is the "data space pointer". It points to the MOPS, or to the internal memory buffer containing the data in MOPS format.
- **viewname**, **graphname**, **plotname** are all of type (char *). They represent the names of views, graphs, plots, and should point to character arrays of max. length DV_NAMEL (=16).

The functions are divided in classes by their functionality, and inside each class they are listed in order of importance.

IMPORTANT : Before getting discouraged by the huge number of routines described in this chapter, the reader should realize that it is possible to get started using just the following few routines :

MOPS ROUTINES (fully explained in Appendix 4)

- **c_sdalloc** : to create a MOPS
- **c_sdacc** : to access a MOPS
- **c_sdini** : to initialize a MOPS
- **c_sdbook** or **c_sdzero** : to create a MOPS object
- **c_sdptr** : to get a pointer to the data part of the MOPS object
- **c_sdquit** : to release access from the MOPS
- **c_sdkill** : to delete a MOPS

XDATAVIEWER ROUTINES (fully explained in this chapter)

- **dv_init** : to initialize the Xdataviewer part of the MOPS
- **dv_vwcreate** : to create a view
- **dv_grcreate** : to create a graph
- **dv_grattach** : to attach a graph to a view
- **dv_plcreate1** or **dv_plcreate2** : to create a plot (1-dim or 2-dim)
- **dv_plattach** : to attach a plot to a graph
- **dv_kick** : to signal the Xdataviewer that new data are available

A simple example is presented in chapter 9, section 1.
All the other routines will let you tailoring properties of views, graphs and plots in the way required by your Application, and implementing sophisticate ways of interacting with the Xdataviewer process.

3.1 Preparation and Configuration of the Xdataviewer

INITIALIZATION OPERATIONS

---

int **dv_init** (int no_views, int no_graphs, int no_plots, char *q)

- **no_views** is the maximum number of views foreseen by the programmer
- **no_graphs** is the maximum number of graphs foreseen by the programmer
- **no_plots** is the maximum number of plots foreseen by the programmer

This routine creates inside the data space pointed by *q* the Xdataviewer objects DV_$VIEW, DV_$GRAPH, DV_$PLOT, needed to store the information related to views, graphs and plots subsequently created. These objects are dimensioned accordingly with the arguments passed to the routine,

and all the slots are marked as “empty”. Other two objects are created : DV_$COMMUNICATE, used for interactions between the Application Program and the Xdataviewer, and DV_$HELP, to contain the name of the *help* file.
This routine has to be called before any view, graph or plot can be created.
For information about how much memory in the data space will be used for the allocation of views, graphs and plots, use *dv_space*.

return value : DV_OK or DV_NO_ROOM

---

int **dv_space** (int no_views, int no_graphs, int no_plots, int *space_needed)

- **no_views** is the maximum number of views to be created
- **no_graphs** is the maximum number of graphs to be created
- **no_plots** is the maximum number of plots to be created
- **space_needed** is the returned value specifying the number of bytes needed to store the 5 Xdataviewer objects.

This routine is useful when estimating the size requirements for the MOPS. See a more detailed explanation in chapter 8 (“Hints & Tricks”).

return value : 0

---

int **dv_delentry** (int entrytype, char *entryname, char *q)

- **entrytype** is one of DV_VIEWTYPE, DV_GRAPHTYPE, DV_PLOTTYPE
- **entryname** is the name of the view, graph or plot to be deleted

This routine deletes the named object from the data space. Alternative to *dv_vwhide*, *dv_grdetach*, *dv_pldetach*.

return value : DV_OK, DV_NO_SUCH_PLOT, DV_NO_SUCH_GRAPH , DV_NO_SUCH_VIEW or DV_NO_SUCH_TYPE

---

int **dv_set_help** (char *filename, char *q)

- *filename* is the name of the help file to be used by the Xdataviewer

The Help Facility is described in more detail in chapter 11. As a reminder, the Help File is an ASCII file, in which normal text lines are mixed with special lines identifying views, graph, plots or select options. Its goal is to provide the User with an online explanation about the meaning of the data displayed by the Xdataviewer.
Warning : calling *dv_init* after *dv_set_help* will delete the name of the Help File.

return value : DV_OK or DV_NO_ROOM

---

int **dv_set_view** (char *viewname, char *q)

- **viewname** is the name of the new view to be displayed by the Xdataviewer

View *viewname* will be displayed as soon as the Xdataviewer is signalled (eg *dv_kick*) that new data has to be shown. If *viewname* does not exist, an error will be generated when the Xdataviewer tries to display the data.

return value : DV_OK or DV_NO_ROOM

---

int **dv_set_view_and_graph** (char *viewname, char *graphname, char *q)

- **viewname** is the name of the new view to be displayed by the Xdataviewer
- **graphname** is the name of the graph (attached to *viewname*) to be displayed by the Xdataviewer

The subview corresponding to *graphname* of view *viewname* will be displayed as soon as the Xdataviewer is signalled (eg via *dv_kick*) that new data has to be shown. If *viewname* or *graphname* do not exist, or if *graphname* is not attached to *viewname*, an error will be generated when the Xdataviewer tries to display the data.

return value : DV_OK or DV_NO_ROOM

ROUTINES MODIFYING THE XDATAVIEWER BEHAVIOR

---

int **dv_set_update_mode** (int mode, char *q)

- **mode** is one of DV_UPDATE_VIEW, DV_UPDATE_DATA, DV_UPDATE_POINTS, DV_UPDATE_ONE_POINT

The value of *mode* defines the strategy used by the Xdataviewer when updating the displayed data.

- DV_UPDATE_VIEW : in this case, the view will be re-displayed from scratch, with its initial zooms and subviews.
- DV_UPDATE_DATA : zooms and subviews currently set by the User will be preserved, and only the data points will be redrawn.
- DV_UPDATE_POINTS : in this case, also the existing data points will be left unchanged, and eventual new points will be added to the plot. This could save time, especially in case of large plots.
- DV_UPDATE_ONE_POINT : only one new point will be added. This mode and the previous one should not be used if graphs using the automatic scaling are involved.

return value : DV_OK or DV_NO_ROOM

---

int **dv_set_raising_mode** (int mode, char *q)

- **mode** is one of DV_NORAISE (default), DV_AUTORAISE

If mode is set to DV_AUTORAISE, every time the Xdataviewer is signalled that new data is available for display, it will raise its own window to become completely visible.

return value : DV_OK or DV_NO_ROOM

ROUTINES RELATED TO SAVING THE CONTENTS OF A VIEW BACK INTO THE DATA SPACE
(see also *dv_kick_and_save*)

---

int **dv_set_save_request** (char *q)

If this routine is used, when the Xdataviewer receives the signal to update the displayed data, it automatically writes back into the data space the contents of the data objects associated with the plots displayed in the view.

return value : DV_OK or DV_NO_COMM

---

int **dv_status_set** (int option, int value, char *q)

- **option** corresponds to the setting to be modified (DV_AUTOSAVE or DV_VIEWCHANGED)
- **value** is the new value (0 / 1) for the specified setting

If the contents of the current view have changed due to editing operations, the Xdataviewer normally asks the User if the view has to be saved back into the data space, before selecting another view for display. Using the *dv_status_set* routine this behaviour can be modified :

If *option* == DV_AUTOSAVE and *value* == 1, modified views will be automatically saved without asking for User's confirmation.

If *option* == DV_AUTOSAVE and *value* == 0, the automatic saving of the views will be again switched off (default behaviour)

If *option* == DV_VIEWCHANGED and *value* == 1, the Xdataviewer will think that the view has been modified, even if this was not the case.

If *option* == DV_VIEWCHANGED and *value* == 0, the Xdataviewer will think that the view has not been modified, even if this was the case.

return value : DV_OK, DV_NO_COMM, DV_ILLEGAL

---

int **dv_status_get** (int option, char *q)

- **- option** is one of DV_AUTOSAVE, DV_VIEWCHANGED

The routine will return the current value of the specified option.

return value : The value required (>=0), or DV_NO_COMM, DV_ILLEGAL

ROUTINES ACTING ON THE XDATAVIEWER GRAPHICAL INTERFACE

---

int **dv_mode_set** (int option, int value, char *q)
int **dv__mode_set** (int option, int value) (for Embedded Dataviewer)

- **- option** corresponds to the setting to be modified (DV_SET_ZOOMTYPE or DV_SET_OP_MODE)
- **- value** is the new value for the specified setting.

This routine enables the programmer to modify, from the program, the Zoom Type or the Operation Mode setting in the Xdataviewer Graphical Interface.
If *option* == DV_SET_ZOOMTYPE, *value* can be one of DV_ZOOM_BOX, DV_ZOOM_HORIZONTAL, DV_ZOOM_VERTICAL.
If *option* == DV_SET_OP_MODE, *value* can be either DV_OP_ONE or DV_OP_ALL.

return value : DV_OK, DV_NO_COMM or DV_ILLEGAL.
Warning : in case of data coming from a Unix Shared Memory MOPS and standalone Xdataviewer, if the routine returns DV_OK, q is no longer valid after the call (the data space needs to be reattached).
Note for Embedded Dataviewer Users : this routine can only be called once the Xdataviewer Interface has been created.

---

int **dv_set_external_menu_string** (int option, char *optionstring, char *q)

- **- option** is one od DV_WRT, DV_SIG, DV_WRT_AND_SIG, DV_STOP_SIG or DV_START_SIG
- **- optionstring** is a character string, max 15 characters

Via this routine the programmer can change the names of the first 5 options in the “External Menu”, used for communication between the Xdataviewer and the Application. If *optionstring* is empty, the default string will be used. If *optionstring* consists of a single space, the corresponding option will be disabled. The default names for these 5 options are, respectively, “Write_Data”, “Write_Data_Send_Signal", “Send_Signal”, "Stop_Incoming_Signal", “Start_Incoming_Signal”.

return value : DV_OK or DV_NO_ROOM

3.2 Interaction with the Xdataviewer

The interaction between the Application Program and the standalone version of the Xdataviewer is based on Unix signals. The Application can signal the Xdataviewer that new data is ready for display (*dv_kick*, or *dv_kick_and_save*), and the Xdataviewer can signal the Application, to inform it that a plot has been edited, or that the User has selected one of the Application dependent options (“Select” Menu, config-

ured via routines *dv_set_selection_menu* and *dv_set_selection_prompt*), or a data point in a plot. The Application can use two routines (*dv_get_selection* and *dv_get_selection_cursor*) to get a complete information about why it was signalled and which data point was selected by the User.
In the case of the Embedded Dataviewer, routine *dv_kick* has been replaced by *dv__kick*, which invokes the data read and redraw routine without the need of a signal (here the Dataviewer is embedded in the same process as the Application). The signalling of the Application by the Embedded Dataviewer is still implemented using a Unix signal.

---

int **dv_kick** (char *q) for standalone Xdataviewer
int **dv__kick**() for Embedded Dataviewer

dv_kick gets the Xdataviewer Process Id from the data space pointed by *q*, and sends a Unix signal SIGUSR1 (=16) to that process. The Xdataviewer reaction to this interrupt will be to reread the currently displayed view from the data space, and to redraw it. This will be done unless the Xdataviewer is already occupied doing that, in which case the signal will be ignored. If the data space is a Unix shared memory MOPS, it will be released, so that *q* will become meaningless after the call, and the data space will have to be reattached by the application process.
In case the Xdataviewer Process Id could not be found in the data space, an error will be returned.
In the case of the Embedded Dataviewer, the routine *dv__kick* will invoke directly the reread and redraw routine belonging to its same process, so that no signal is needed. Also, the data space will not be released.
See also routine *dv_kick_file*, in case the data space pointed by *q* was loaded from a file.

return value : DV_OK or DV_NO_PID

---

int **dv_kick_and_save** (char *q) for standalone Xdataviewer
int **dv__kick_and_save**() for Embedded Dataviewer

Same principle as *dv_kick*, but the Xdataviewer will also save the current view back into the data space before rereading and redrawing it.

return value : DV_OK or DV_NO_PID

---

int **dv_get_dv_processid** (int *dv_processid, *char *q)

- **dv_processid** is the (returned) process identifier of the Xdataviewer attached to the data space

This routine will return the Process Id of the Xdataviewer attached to the data space pointed by *q*. It will also check for the existence of the process corresponding to the Process Id found.

return value : DV_OK or DV_NO_PID

---

int **set_dv_app_processid** (int app_processid, char *q)

- **app_processid** is the Process Id of the application

Writes into the specific field of the data space the process identifier of the Application, so that the Xdataviewer can send a signal to it when required. Typically the *app_processid* should be the process identifier of the calling program, but if the Application Program wants to disable the signalling capabilities of the Xdataviewer, it could set this argument to 0 . Then the Xdataviewer will never signal the Application (until *set_dv_app_processid* is called again with the right value).

return value : DV_OK or DV_NO_ROOM

---

int **dv_set_selection_menu** (int noptions, char options[][16])

- **noptions** is the number of Application dependent options to be inserted in the Xdataviewer Select List (max. 50 options can be defined).
- **options** is the list of the option strings. Each option is a text string of max. 15 characters.

Sets the list of the Application dependent options. The list will be popped up when the Xdataviewer User clicks on the Select Icon of the Xdataviewer Graphical Interface. If then an option of the list is selected, the Application will be notified of the selection via a Unix signal.

return value : DV_OK or DV_NO_ROOM

---

int **dv_set_selection_prompt** (int noption, char *prompt, char *q)

- **noption** is the index of one of the Application dependent options
- **prompt** is a text string (max. 15 characters) .

Sets the instructions (ex.“*pick a point*”) for the Xdataviewer User, to tell him what to do after he has chosen the specified Application dependent option. The message will be displayed in the Xdataviewer User Info Message space when the corresponding option is selected.

return value : DV_OK or DV_NO_ROOM

---

int **dv_get_selection** (int *option, char *viewname, char *graphname, char *plotname, char *xdataname, char *ydataname, char *textname, int *index, double *xvalue, double *yvalue, char *textstring, char *q)

- **option** (returned) : the Application dependent option selected by the Xdataviewer User
- **viewname** (returned) : name of the selected view, if any. (16 chars)
- **graphname** (returned) : name of the selected graph, if any. (16 chars)
- **plotname** (returned) : name of the selected plot, if any. (16 chars)
- **xdataname** (returned) : name of the x object of the selected plot, (if any). (80 chars)
- **ydataname** (returned) : name of the y object of the selected plot, (if any). (80 chars)
- **textname** (returned) : name of the text label object of the selected plot, (if any). (80 chars)
- **index** (returned) : index to the data element selected, if any, or -1
- **xvalue** (returned) : xdataname[index], if it exists
- **yvalue** (returned) : ydataname[index], if it exists
- **textstring** (returned) : textname[index], if it exists. (16 chars)

This routine returns information about the last item selected within the Xdataviewer. When an Application dependent option is selected (from the Select List), a signal is sent immediately and *index* will be set to *-1*, other items will be null. If the signal has not been sent as a result of a Select operation, then *option* will be either -DV_SIG or -DV_WRT_SIG . The programmer should pass as arguments character strings that can contain the returned values.

return value : DV_OK or DV_NO_SELECTION.

---

int **dv_get_selection_cursor** (double *x, double *y, char *q)

- **x** (returned) : cursor x position
- **y** (returned) : cursor y position

This routine returns the position of the cursor when the last selection was made. It is really an appendix to *dv_get_selection* .

return value : DV_OK or DV_NO_SELECTION

FORCING XDATAVIEWER TO REPLACE THE DATA AREA SPACE WITH A NEW ONE

The following function will be used by an Application Program to force the associated Xdataviewer to display a new data area space. At the implementation level, this is achieved by depositing the name of the new data area space in a dedicated field of the old one, and then informing the Xdataviewer about that. This routine works in combination with *dv_kick*, or *dv_save_file* and *dv_kick_file*, as shown in example 9.8 .

int **dv_newdata_set** (char *newdata_name, char *q)

- **newdata_name** is the name of the new data area space to be displayed by the Xdataviewer. If its first character is '*', *newdata_name* will be used as a file name, otherwise as a MOPS name .
- **q** is the usual pointer to the currently used data area space, a dedicated field of which will receive the *newdata_name* string. If the current data area space is a file, *q* is meaningless.

This routine deposits in a dedicated field of the current data area space the name of a new data area space to be used, and sets a flag which the Xdataviewer will interpret (when signalled) as a request of data area space replacement. To activate the change, the Application program must also call *dv_kick* (if the current data area space was a MOPS) , or *dv_save_file* and *dv_kick_file* (if it was a MOPS-formatted file).

return value : 0 (no check on the validity of the new data area space is performed)

3.3 Operations with a Mops contained in binary files

The latest Xdataviewer release also provides some routines to configure the data space by loading files containing the binary representation of a MOPS. Using the following routines the Application can produce such a file, reload it, modify it, communicate with the Xdataviewer. A typical usage of these routines is when the data space, of the size of a few hundreds kilobytes, does not need to be refreshed at a real-time speed. In this case working with binary files will avoid the need to create a Unix shared memory, which should be deleted at a later stage to clean the computer Operating System resources. One of the examples in chapter "Examples" will show the proper way for using these routines.

int **dv_save_file** (char *filename, char *q)

- **filename** is the file where the current data space (pointed by *q*) has to be saved

This routine writes into file *filename* the binary representation of the data space pointed by *q*. The file can then be reloaded by the Application and by the Xdataviewer.

return value : 0 (success) or -1 (failure)

char ***dv_attach_file** (char *filename)

- **filename** is the file containing the binary representation of the MOPS to be reloaded.

If the file can be read, the routine allocates (*malloc*) a memory buffer large enough to contain the file size, and does a binary read from the file to the buffer. The routine returns the pointer to the allocated buffer. It is responsibility of the Application to release the memory space (by *dv_quit_file*) when it is not needed anymore.

return value : pointer to the allocated buffer (success) or -1 (failure)

char ***dv_attach_filegrow** (char *filename, int size, int mode)

- **filename** is the file containing the binary representation of the MOPS to be reloaded.
- **size** is the additional or absolute size of the space to be reserved.
- **mode** is one of DV_ABSOLUTE_SIZE, DV_RELATIVE_SIZE

This routine is similar to the previous one, with the difference that the memory buffer allocated will contain some additional space, needed for example if data objects have to increase in size. If *mode* = DV_ABSOLUTE_SIZE, *size* will define the size of the allocated buffer (the file size will still be used if *size* is too small). If *mode* = DV_RELATIVE_SIZE, *size* will be added to the file size.

return value : pointer to the allocated buffer (success) or -1 (failure)

int **dv_save_filecomm** (char *filename, char *q)

- **filename** is the file where the DV_$COMMUNICATE object has to be saved

Mainly used internally, this routine updated just the DV_$COMMUNICATE object, used for communication between the Application and the Xdataviewer, inside the file *filename*.

return value : 0 (success) or -1 (failure)

---

int **dv_quit_file** (char *q)

Frees the memory buffer previously allocated. After this call *q* is no longer valable (the file needs to be reattached).

return value : 0

---

int **dv_kick_file** (char *q)

Equivalent to *dv_kick*. It frees the memory buffer and sends a signal to the Xdataviewer, if it can find its Process Id. After this call *q* is no longer valid (the file needs to be reattached).

return value : DV_OK or DV_NO_PID

---

int **dv_set_enablewriting** (int mode, char *q)

- **mode** is one of DV_DISABLEWRITING, DV_ENABLEWRITING

By default, the Xdataviewer will not be able to overwrite a file which it has loaded. This is because one of the goals of saving a MOPS into a file is to provide a easy way for saving data prepared by an Application in a format immediately displayable by the User. In this case, the file should not be modified. On the other hand, if one wants to use these routines to perform communication between the Xdataviewer and the Application, the Xdataviewer must be able to rewrite the file. The Application will grant this right to the Xdataviewer by calling *dv_set_enablewriting* with *mode* = DV_ENABLEWRITING. This will modify a flag in the DV_$COMMUNICATE object of the data space. The data space has then to be saved into a file (for example by *dv_save_filecomm*), and the Xdataviewer kicked (*dv_kick_file*). The Xdataviewer will then reread the file, and will find in it the writing permission. If *mode* = DV_DISABLEWRITING, the writing permission will be removed.

return value : DV_OK or DV_NO_ROOM

3.4 Default setting routines

At their creation, Xdataviewer graphs and plots are given a default setting for many properties (e.g colour, marker type). These settings can be individually modified using the routines described in chapters “Graphs” and “Plots”. Often this is not very practical, especially when many different graphs and plots have to be modified in a similar way. To overcome this problem, we have introduced a few routines which act on the default settings. Whenever a default setting is modified by one of these routines, the new value will become the default to be used for all graphs and plots created after the modification.
Three routines are necessary, because of the different types of default values.

---

int **dv_set_default** (char *name, long value)

- **name** is the name of the parameter for which the default has to be changed
- **value** is the intended new value for the default setting.

Parameters involved in PLOT creation

If name == “HISTCOL” value will be used as the default value for histogram colour.

If name == “LINECOL” value will be used as the default value for line colour.

If name == “MARKCOL” value will be used as the default value for marker colour.

If name == “MARKTYP” value will be used as the default value for marker type.

If name == “COLOURED_LINE”, value will be used to determine the strategy for colouring lines of plots of type DV_LINE.

- If value == 0 the colour of the line in a plot of type DV_LINE will be decoupled from the colours specified in the colour object for the plot (see also *dv_plcolouredline*) .

- If value == 1 , also the line will use the colours defined in the colour object.

If name == “DATA_FORM”, value (DV_XMONOTONIC, DV_RANDOM or DV_XUNIFORM) will determine the strategy used by the Dataviewer when displaying a plot and the associated cursor.

If name == “READ_WRITE”, value (DV_READ or DV_WRITE) will determine if a plot is editable by default, or not.

If this routine is not used, the default values are respectively DV_WHITE, DV_WHITE, DV_WHITE, DV_DOT , 1 , DV_XMONOTONIC, DV_READ.

Parameters involved in GRAPH creation

If name == “GRID”, value (DV_GRID_OFF, DV_GRID_ON, DV_GRID_FULL) will be used as the default grid value for dataviewer graphs.

If name == “ZEROLINE”, value (DV_ZEROLINE_OFF, DV_ZEROLINE_ON, DV_ZEROLINE_XY) will be used as the default zeroline value for dataviewer graphs.

If name == “XAXSCALE”, value (DV_LINEAR, DV_LOG, DV_TIME) will be used to define the default horizontal scale for a graph.

If name == “YAXSCALE”, value (DV_LINEAR, DV_LOG) will be used to define the default vertical scale for a graph.

If name == “GRAPH_TYPE”, value (0,1,100 or 101) will determine if by default a graph should be listed and merged, or not.

If this routine is not used, the default values are respectively DV_GRID_OFF, DV_ ZEROLINE_OFF, DV_LINEAR, DV_LINEAR, 0 .

return value : 0 if name is legal, -1 if it does not correspond to one of the above mentioned cases.

---

int **dv_set_defaultlimit** (char *name, double value)

- **- name** is the name of the parameter to be changed
- **- value** is the intended new value for the default setting.

Used to modify graph limits (routines *dv_grlimits* and *dv_grpercent*)

If name == “XLIMIT” (“YLIMIT”) , value will be used as a margin to be left on the X (Y) axe of a graph, around the space used to display the data.

If name == “XMIN” , “XMAX”, “YMIN”, “YMAX”, value will be used as a default limit for the correspondent coordinate in a graph. If XMIN >= XMAX (YMIN >= YMAX) the X (Y) limit will be modified dynamically, to accommodate all the data.

Default values are XLIMIT = YLIMIT = 5.0, XMIN = XMAX = YMIN = YMAX = 0.0

return value : 0 if name is legal, -1 if it does not correspond to one of the above mentioned cases.

---

int **dv_set_defaultformat** (char *name, char *value)

- **- name** is the name of the parameter to be changed
- **- value** is the intended new value for the default setting.

Routine used to set the default formats to be used when listing a plot.

If name == “X”, value will be used as format for the X object associated with the plot.

If name == “Y”, value will be used as format for the Y object associated with the plot.

Default values are the empty strings “”, “”, which will let the Dataviewer deciding which format to use.

return value : 0 if name is legal, -1 if it does not correspond to one of the above mentioned cases.

---

int **dv_show_defaults**()

This routine prints all the default values currently set by the previous functions.

return value : 0

3.5 Views

VIEW CREATION

---

int **dv_vwcreate** (char *viewname, char *viewtitle, char *menuentry, int no_rows, int no_cols, char *q)

- **viewname** is the name of the view to be created. Max. 15 characters.
- **viewtitle** is a text string (max 80 characters) to be displayed at the left top corner of the view.
- **menuentry** is a text string (max 80 characters) used to identify the view in the view list.
- **no_rows** is the number of rows for the layout of graphs inside this view.
- **no_cols** is the number of columns for the layout of graphs inside this view.

It creates a view “*viewname*” and sets its *title* (to be displayed on the screen when the view is selected) and *menuentry* (to appear in the list of the views, from which the User can select one at a time). The view will lay out all the graphs attached to it (max 16, use *dv_grattach*) according to *no_rows* and *no_cols*, and to the values specified in the *dv_grattach* calls. If *no_rows* or *no_cols* is zero, a default layout will be used.

return value : DV_OK, DV_NO_ROOM or DV_VIEW_EXISTS

---

int **dv_vwchnames** (char *viewname, char *viewtitle, char *menuentry, char *q)

Same arguments as for *dv_vwcreate*. It enables the programmer to change *title* and *menuentry* for the view.

return value : DV_OK or DV_NO_SUCH_VIEW

---

int **dv_vwchtitle** (char *viewname, char *viewtitle, char *q)

Same arguments as for *dv_vwcreate*. It enables the programmer to change just the *title* for the view.

return value : DV_OK or DV_NO_SUCH_VIEW

VIEW AVAILABILITY

The two following routines enable the programmer to temporarily hide the existence of a view to the User, without deleting the view itself. This can be very convenient when many views have been defined, making the View List long and confuse, and at a particular stage of an application only a subset of these views contain useful information.

---

int **dv_vwhide** (char *matchname, int mode, char *q)

- **matchname** is a character string to match the name of the view(s) to be hidden.
- **mode** determines the way of comparing *matchname* with the view names.

Goal of this routine is to remove from the View List one or more views, making them unselectable by the User. The views which match *matchname* according to the mode specified will be hidden from the View List, until an equivalent call to *dv_vwshow* is effectuated.

- - If *matchname* = #ALL# , all the views will be hidden
- - If *mode*=0, the routine applies only to a view with name identical to *matchname*
- - If *mode*=1, the routine applies to all views whose names start by *matchname*
- - If *mode*=2, the routine applies to all views whose names contain the string *matchname*
- - If *mode*=3, the routine applies to all views whose names end by *matchname*

return value : DV_OK or DV_NO_SUCH_VIEW

---

int **dv_vwshow** (char *matchname, int mode, char *q)

Same arguments as for *dv_vwhide*. This routine reverses the effect of *dv_vwhide*. The views matching the match criteria will be shown again in the View List.

return value : DV_OK or DV_NO_SUCH_VIEW

3.6 Graphs

GRAPH CREATION AND ATTACHEMENT

---

int **dv_grcreate** (char *graphname, char *title, char *xlabel, char *ylabel, char *q)

- - **graphname** is the name of the graph to be created. Max 15 characters.
- - **title** is a text string (max 80 characters) to be displayed at the left top corner of the graph.
- - **xlabel** is a text string (max 80 characters) to be used as label for the x-axis of the graph.
- - **ylabel** is a text string (max 80 characters) to be used as label for the y-axis of the graph.

It creates a graph "*graphname*", and sets *title*, *xlabel* and *ylabel* for the graph. The graph should be attached to one or more views using *dv_grattach*. Up to 16 plots can be attached to the graph, using *dv_plattach*.

return value : DV_OK, DV_NO_ROOM, DV_GRAPH_EXISTS

---

int **dv_grattach** (char *graphname, char *viewname, int rowpos, int colpos, int rowsize, int colsize, char *q)

- - **graphname** is the name of the graph to be attached
- - **viewname** is the name of the view to which the graph has to be attached
- - **rowpos** is the start row of the graph
- - **colpos** is the start column of the graph
- - **rowsize** is the size of graph in rows
- - **colsize** is the size of graph in columns

Attaches the graph "*graphname*" to the view "*viewname*" (both should have been previously created). By default, all graphs will be the same size, and will be positioned on the screen according to the "*no_rows*" and "*no_cols*" specified in *dv_vwcreate*. In this case, a value of 0 shall be used for *rowpos*, *colpos*, *rowsize*, *colsize*. If specific positioning and sizing are required, then *rowpos*, *colpos*, *rowsize* and *colsize* must be correctly set for all the graphs attached to the view. Their values are relatives to the "*no_rows*" and "*no_cols*" given in *dv_vwcreate*.
A graph can be attached to as many views as wished, but a view cannot have more than 16 graphs attached to it.

return value : DV_OK, DV_NO_ROOM, DV_NO_SUCH_GRAPH, DV_GRAPH_EXISTS or DV_NO_SUCH_VIEW

---

int **dv_grdetach** (char *graphname, char *viewname, char *q)

Detaches the graph *"graphname"* from the view *"viewname"*.

return value : DV_OK , DV_NO_SUCH_GRAPH, or DV_NO_SUCH_VIEW.

SETTING AND MODIFYING GRAPH PROPERTIES

---

int **dv_grchnames** (char *graphname, char *title, char *xlabel, char *ylabel, char *q)

Same arguments as for *dv_grcreate*. It resets the *title*, *x axis label* and *y axis label* associated with graph "*graphname*" .

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grlabels** (char *graphname, int nlabels, char labels[][80], char *q)

- **- graphname** is the name of the graph
- **- nlabels** is the number of labels (max. 5)
- **- labels** is an array of label strings (max. 80 char. each)

Via this routine, the programmer is able to define 5 lines of text which will be displayed near the top left hand corner of the graph.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grlimits** (char *graphname, double xmin, double xmax, double ymin, double ymax, char *q)

- **- graphname** is the name of the graph
- **- xmin** is the minimum x coordinate
- **- xmax** is the maximum x coordinate
- **- ymin** is the minimum y coordinate
- **- ymax** is the maximum y coordinate

Can be used to define the boundaries of the window on the x,y plane visible through the graph (by default, the boundaries are computed to accommodate all the data points of the plots attached to the graph). If *xmin >= xmax*, the x boundaries will still be computed from the data points. If *ymin >= ymax*, the y boundaries will still be computed from the data points.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grpercent** (char *graphname, double xpercent, double ypercent, char *q)

- **- graphname** is the name of the graph
- **- xpercent** is the margin (expressed as percentage of the x data range) to be added to the graph x boundaries computed to accommodate all the data points.
- **- ypercent** is the margin (expressed as percentage of the y data range) to be added to the graph y boundaries computed to accommodate all the data points.

By default, a margin of 5 % of the data range is added to the boundaries computed to accommodate all the data points. This routine can change this default value, independently in x and y.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grscales** (char *graphname, int xscale, int yscale, char *q)

- **- graphname** is the name of the graph

- **- xscale** is one of DV_LINEAR, DV_LOG, DV_TIME
- **- yscale** is one of DV_LINEAR, DV_LOG

The scaling for each axis of the graph can be independently set to linear or logarithmic. The x axis scaling can be also set to DV_TIME. In this case, the plots attached to the graph should be 2-dim plots, with Unix timestamps as the elements of the x data object, as in the following example :

```
.....
/* define mops objects with data and timestamps (common for the two data sets)*/
c_sdbook(MAX_DATA,4L,"DATA_A","float",q);
c_sdbook(MAX_DATA,4L,"DATA_B","float",q);
c_sdbook(MAX_DATA,4L,"timestamp","long",q);
.....
/* create a graph , and set its X scale of type DV_TIME */
dv_grcreate("mygraph","graphlabel","time","data",q);
dv_grscales("mygraph",DV_TIME,DV_LINEAR,q);
.....
/* create the plots. The attachments to the graph are omitted */
dv_plcreate2("plot_a",DV_MARKER,"timestamp","DATA_A",q);
dv_plcreate2("plot_b",DV_MARKER,"timestamp","DATA_B",q);
.....
```

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grsetgrid** (char *graphname, int grid, char *q)

- **- graphname** is the name of the graph
- **- grid** specifies the grid type to be used for the graph (DV_GRID_OFF, DV_GRID_ON, DV_GRID_FULL)

A self-adaptive grid can be drawn around a graph. By using this routine the programmer can determine whether or not the grid has to be drawn when the graph is initially displayed. DV_GRID_OFF means no grid, DV_GRID_ON will generate a set of horizontal and vertical tics around the graph, DV_GRID_FULL a grid with horizontal and vertical lines across the graph display.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grsetzeroline** (char *graphname, int zeroline, char *q)

- **- graphname** is the name of the graph
- **- zeroline** specifies the type of axis to be drawn (DV_ZEROLINE_OFF, DV_ZEROLINE_ON, DV_ZEROLINE_XY)

It is also possible to draw the x and y axis on a graph. This routine can be used to determine the initial state for the graph (respectively, no axis drawn, only x-axis, x- and y-axis drawn).

return value : DV_OK or DV_NO_SUCH_GRAPH

LISTING A GRAPH IN TEXT FORMAT

The old release of the Dataviewer only allowed the listing of a plot at a time. For many applications it is much more convenient to be able to list (and edit) all the plots in a graph at the same time. This involves an additional complication when the plots in a graph have different x coordinate arrays. In this case, the Xdataviewer will have to produce an ordered superset of x coordinates, including all the x values present in at least one plot . The x values of the superset will be printed line by line, together with the corresponding y values for the plots where a point with that x coordinate can be found, and with the symbol "..." for plots which do not have such x-coordinate. In order to trigger this behaviour, routine

dv_grlistmerge will have to be used.
WARNING : This method does not work if the plots are not monotonically increasing in x.

---

int **dv_grlist** (char *graphname, int listopt, char *q)

- **graphname** is the name of the graph
- **listopt** is one of DV_LIST (text format), DV_NOLIST (graphic, default)

Using this routine, the programmer can determine the initial mode (text of graphic) in which the graph is shown.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grlistmerge** (char *graphname, int listopt, char *q)

- **graphname** is the name of the graph
- **listopt** is one of DV_MERGE (enables listing of inhomogeneous graphs) , DV_NOMERGE

This routine has to be invoked with *listopt* = DV_MERGE if a graph containing plots with different x coordinate arrays has to be listed in text format.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_grliststyle** (char *graphname, int liststyle, char *q)

- **graphname** is the name of the graph
- **liststyle** is one of DV_LISTFULL (default) or DV_LISTSHORT

When the graph is saved onto a file, if *liststyle* = DV_LISTFULL the values of all the auxiliary objects of all plots will be listed. If *liststyle* = DV_LISTSHORT, only the data objects associated with the plots will be listed.

return value : DV_OK or DV_NO_SUCH_GRAPH

CONTROLLING THE NUMBER OF POINTS TO BE DISPLAYED FOR A GRAPH

---

int **dv_grnpoints** (char *graphname, int npoints, char *q)

- **graphname** is the name of the graph
- **npoints** is the maximum number of points to be displayed

This routine enables the programmer to set a limit to the number of points displayed on the specified graph. If any of the plots attached to the graph has more than *npoints*, only the first *npoints* will be displayed. If *npoints* = -1, no limit will be applied (default).
If a limit is set using this routine, this will have priority over a limit possibly set via *dv_plnpoints*. For an alternative way of limiting the number of points to be displayed, read the discussion of *dv_ObjectResize* in chapter "*Hints & Tricks*", section "*A practical way of dealing with mops objects of variable size*".

return value : DV_OK or DV_NO_SUCH_GRAPH

3.7 Plots

PLOT CREATION AND ATTACHEMENT

---

int **dv_plcreate1** (char *plotname, int plottype, char *ydata, char *q)

- **plotname** is the name of the plot to be created. Max. 15 characters.

- **- plottype** is the type of the plot (one of DV_MARKER, DV_LINE, DV_HISTO, DV_MULTIMARKER, DV_MULTIMARKERLINE). The special plot type DV_PICTURE can be also specified, but it will be described in detail in chapter “A picture drawing facility for the Xdata-viewer”.
- **- ydata** is the name of the mops object containing the y-values for the plot points. The object should be contained in the data space pointed by **q** .

Creates a 1-dimensional plot “*plotname*”, where *ydata*[i] is plotted against *i*. The way in which the plot is shown depends on *plottype* (the default type is DV_MARKER, but it can be modified by routine *dv_set_default*).
If *ydata* does not exist when the plot has to be displayed by the Xdataviewer, an error is generated.
The plot should be associated with one or more graphs using *dv_plattach*.
The Xdataviewer accepts data objects of type short, int, long, float , double .

IMPORTANT : if the first character in ydata is ‘*’, the object will not be contained in a MOPS, but in a file. The remaining part of ydata will be used as a file path name, to specify in which file the object has to be found.
Ex. if ydata = */usr/tmp/junk , the object will be found in file /usr/tmp/junk .
If the filename starts by // , it will be treated as an “URL” location, and searched for on the Web.
Ex. if ydata = *//venice.cern.ch/~opera/junk , it will be searched on the Web as
http:// venice.cern.ch/~opera/junk
This same convention applies to all the objects used in the routines described below.

return value : DV_OK, DV_NO_ROOM, DV_PLOT_EXISTS

---

int **dv_plcreate2** (char *plotname, int plottype, char *xdata, char *ydata, char *q)

- **- xdata** is the name of the mops object containing the x-values for the plot points. The object should be contained in the data space pointed by q .

All the other arguments are the same as for **dv_plcreate1** . This routine creates a 2-dimensional plot, where *ydata*[i] is plotted against *xdata*[i] . If *xdata* or *ydata* do not exist when the plot has to be displayed, or if they have a different number of elements, an error is generated.
The plot should be associated with one or more graphs using *dv_plattach*.
In some particular cases, a DV_HISTO plot created with *dv_plcreate2* will be displayed in a nicer way if routine *dv_pldtform* with argument DV_XUNIFORM is invoked.

return value : DV_OK, DV_NO_ROOM, DV_PLOT_EXISTS

---

int **dv_plattach** (char *plotname, char * graphname, char *q)

- **- plotname** is the name of the plot to be attached.
- **- graphname** is the name of the graph to which the plot has to be attached.

Attaches the plot “*plotname*” to the graph “*graphname*”. Plot and graph should have been previously created . The attachment means that every time the graph is displayed, the plot will be displayed inside the graph. A plot can be attached to as many graphs as wished, but a graph cannot have more than 16 plots attached to it.

return value : DV_OK, DV_NO_ROOM, DV_NO_SUCH_PLOT, DV_PLOT_EXISTS or DV_NO_SUCH_GRAPH

---

int **dv_pldetach** (char *plotname, char * graphname, char *q)

Same arguments as *dv_plattach*. Detaches the plot “*plotname*” from the graph “*graphname*”. Plot and graph should have been previously created and attached .

return value : DV_OK, DV_NO_SUCH_PLOT, DV_NO_SUCH_GRAPH

SPECIFYING THE DATA CHARACTERISTICS FOR A PLOT

---

dv_pldtform (char *plotname, int dataform, char *q)

- **- plotname** is the name of the plot
- **- dataform** is one of DV_RANDOM, DV_XMONOTONIC, DV_XUNIFORM

Depending on the *dataform*, the Xdataviewer chooses its strategy to associate the position of the cursor with one point in a plot. If the x-coordinates of a plot are monotonically increasing, DV_XMONOTONIC can be used, with some gain in processing speed, especially for plots with many points. DV_XUNIFORM can be used to improve the drawing of histogram plots created with *dv_plcreate2*, when the x-coordinates are equally spaced.

return value : DV_OK or DV_NO_SUCH_PLOT

GIVING COLOR TO THE PLOTS

By default, all the points of a plot are drawn using the same colour (white or black, depending on the colour of the Xdataviewer background). It is possible to change the value of this colour, and also to use an independent colour for every point of the plot. The list of the available colours is reported in the appendix.

---

int **dv_plhis** (char *plotname, int histcol, char *q)

- **- plotname** is the name of the plot to be coloured.
- **- histcol** is the colour to be used (DV_BLACK, DV_RED, DV_GREEN, etc.)

If the plot is of type DV_HISTO, each bar of the histogram will be drawn using colour *histcol*, unless function *dv_plcol* is also used.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_pllin** (char *plotname, int linecol, char *q)

- **- plotname** is the name of the plot to be coloured.
- **- linecol** is the colour to be used (DV_BLACK, DV_RED, DV_GREEN, etc.)

If the plot is of type DV_LINE, each point of the plot will be joined by lines with colour *linecol*.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_plmar** (char *plotname, int markercol, int markertype, char *q)

- **- plotname** is the name of the plot to be coloured.
- **- markercol** is the colour to be used (DV_BLACK, DV_RED, DV_GREEN, etc.)
- **- markertype** is the type of marker to be used (DV_POINT, DV_DOT, DV_CROSS, etc)

If the plot is of type DV_MARKER, each point will be represented by a marker with the given *markertype*. The same is true if the plot is of type DV_LINE; in this case, if a markertype DV_NONE is used, no points will be drawn, but only the lines joining them. The colour of the markers will be *markercol*, unless *dv_plcol* is also used.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_plcol** (char *plotname, char *colourname, char *q)

- **- plotname** is the name of the plot to be coloured.
- **- colourname** is the name of the mops object containing the colour values for the plot points. Max. 80 characters.

If the plot contains markers or an histogram, each marker or bar will take its colour from the object *colourname*, eg *xdata*[i] will have *colourname*[i]. If colourname[i] is invalid, the default DV_WHITE will be used. If the object *colourname* has less elements than the data object(s), the Xdataviewer will cycle through the colours restarting from the beginning when the end is reached.
The object *colourname* should be contained in the same data space pointed by *q*.
If the plot is of type DV_MULTIMARKER or DV_MULTIMARKERLINE, the contents of colourname will be used to define a **different marker type** for each point of the plot, and not anymore for colouring it.
An alternative possibility of assigning a different marker type to points in a plot is to add the value of **marker_type_code x 1000** to the corrsponding element of the MOPS colour object. In this way is is possible to **modify both the colour and the marker type** for each point.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_plcolouredline** (char *plotname, int flag, char *q)

- **plotname** is the name of the plot
- **flag** can be 0 or 1

Plotname should be a plot of type DV_LINE . The routine enables the programmer to couple or decouple the colour of the lines joining the plot points from the colour of the points themselves. If *dv_plcol* is used, each point has its own colour. By default, or if *flag* is equal to 1, each line will take the colour of its starting point. If *flag* is set to 0, the lines will be all drawn with the same colour (the default colour, or the one specified by *dv_pllin*).

ASSIGNING LABELS TO PLOT POINTS

---

int **dv_pltxt** (char *plotname, char *text, char *q)

- **plotname** is the name of the plot
- **text** is the name of the mops object containing the labels

dv_pltxt enables the programmer to associate a short text label with each point in a plot. The label will be displayed on the bottom line of the graph when the plot point is the closest to the cursor. It will also be displayed when the plot is listed in text format. The object referred by text must have the same number of elements as the data object(s) for the plot. It should be of type “char”, with element size of 16. Each element will contain the label (max. 15 characters) for the corresponding plot point.

return value : DV_OK or DV_NO_SUCH_PLOT

ASSIGNING ERROR BARS TO PLOT POINTS

---

int **dv_plerrorbars** (char *plotname, char *x_left, char *x_right, char *y_top, char *y_bottom,char *q)

- **plotname** is the name of the plot
- **x_left** is the name of the mops object containing the error values for the horizontal left error bars
- **x_right** is the name of the object containing the error values for the horizontal right error bars
- **y_top** is the name of the object containing the error values for the vertical upwards error bars
- **y_bottom** is the name of the mops object containing the error values for the vertical downwards error bars

This routine enables the programmer to add error bars to the data points of a plot. When the plot is displayed, these errors will be displayed as lines in the 4 directions around the data point. Notice that the four objects used for the error bars must have the same number of elements as the plot, otherwise an error will occur when the plot has to be displayed.

return value : DV_OK or DV_NO_SUCH_PLOT

LISTING A PLOT IN TEXT FORMAT

By default, a plot is initially displayed in graphical format. The User can then list it in text format, acting on the Xdataviewer GUI. The programmer has the possibility of forcing the plot to initially appear in text format, and to control the way in which the plot values are printed.

---

int **dv_pllist** (char *plotname, int listopt, char *q)

- **- plotname** is the name of the plot
- **- listopt** is one of DV_LIST, DV_NOLIST

If *listopt* is set to DV_LIST, the plot will be initially displayed in text format. Otherwise the graphical representation will be used. If this option is set for more than one plot in a graph, the one which is listed will usually be the last plot created (to simultaneously list all the plots in a graph use *dv_grlist*, or option *list_graph* from Xdataviewer GUI).

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_plcoltxt** (char *plotname, char *colourname, char *q)

- **- plotname** is the name of the plot
- **- colourname** is the name of the mops object containing the colour values for the plot points. Max. 80 characters.

Equivalent to *dv_plcol*, it enables the programmer to associate a different colour with every point of a plot. These colours, contained in the mops object specified by *colourname*, will be used when the plot is listed in text format. If the colour object contains less elements than the plot, the Xdataviewer will cycle more than once through the colours.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_plprecision** (char *plotname, int precision_x, int precision_y, char *q)

- **- plotname** is the name of the plot
- **- precision_x** specifies the number of digits after the decimal point to be used when listing the x-coordinates of the plot.
- **- precision_y** specifies the number of digits after the decimal point to be used when listing the y-coordinates of the plot.

Alternative to the next function. It gives the programmer control over the output format of the text representation of the plot. *precision_x* and *precision_y* should be between 0 and 14, or -1 if the Xdataviewer default format is to be used.

return value : DV_OK, DV_NO_SUCH_PLOT, or DV_OUT_OF_RANGE

---

int **dv_pllistform** (char *plotname, char *format_x, char *format_y, char *q)

- **- plotname** is the name of the plot
- **- format_x** specifies the output format to be used when listing the x-coordinates of the plot. Max. 8 characters.
- **- format_y** specifies the output format to be used when listing the x-coordinates of the plot. Max. 8 characters.

Gives the programmer even more control than the previous one.

return value : DV_OK, DV_NO_SUCH_PLOT, or DV_OUT_OF_RANGE

HOW TO MAKE A PLOT EDITABLE

By default, plot are readonly. That means that the possibility of editing a plot using the Xdataviewer

Graphical Interface is usually disabled. Only by explicitly specifying that the plot can be edited, the Editing Facilities are re-enabled.

---

int **dv_plreadwrite** (char *plotname, int readwrite, char *q)

- **- plotname** is the name of the plot
- **- readwrite** is either DV_READ or DV_WRITE

If *readwrite* is set to DV_READ (default), the plot cannot be edited. If *readwrite* is set to DV_WRITE, the plot can be edited. The programmer can "or" DV_WRITE with DV_ENDPT_FIX_X to prevent moving the plot endpoints horizontally, or with DV_ENDPT_FIX to prevent any modification to the endpoints.

return value : DV_OK, or DV_NO_SUCH_PLOT

CONTROLLING THE NUMBER OF POINTS TO BE DISPLAYED FOR A PLOT
By default, all the points in a plot (defined by the number of elements of the mops data object(s) specified at the plot creation) are shown when the plot is displayed, at the condition that they lie in the x,y visible window defined by the graph containing the plot. In some cases it may be useful to show only a subset of the plot. This can be done by the following routine. There are also other ways of obtaining a similar behaviour, and the one which I recommend, whenever possible, is described in chapter "*Hints & Tricks*", section "*A practical way of dealing with mops objects of variable size*" (routine *dv_ObjectResize*)

---

int **dv_plnpoints** (char *plotname, int npoints, int first, char *q)

- **- plotname** is the name of the plot
- **- npoints** is the number of points to be displayed
- **- first** is the index of the first point to be displayed

Only the number of points specified by *npoints*, starting at the position specified by *first*, will be displayed. Notice that this routine does not override the behaviour obtained by a call to *dv_grnpoints*. So, if the plot has to be displayed within a graph for which *dv_grnpoints* has been used, the graph constraints will have priority over the plot ones.

return value : DV_OK, or DV_NO_SUCH_PLOT

3.8 Objects

---

int **dv_ObjectResize** (char *objectname, int no_elements, char *q)

- **- objectname** is the name of the MOPS object
- **- no_elements** is the number of elements that the Xdataviewer will think the object has.

This routine modifies the "number of element" field in the MOPS directory entry corresponding to *objectname* . *no_elements* must be smaller than the number of elements initially allocated for the object. In this way the Xdataviewer will believe that the plot has a number of points equal to *no_elements*.
In case of 2-dim plots, or when auxiliary objects like errorbars are used, all these objects must be "resized" to the same number of elements. See also a discussion in chapter "Hints & Tricks".

WARNING : THIS ROUTINE SHOULD BE USED WITH CAUTION. In particular, it should not be used if the data is going to be edited by the Xdataviewer. It should only be used if the sizes of all the objects in the MOPS are initialized once for all (*c_sdbook* or *c_sdzero*), and never modified with *c_sdcut*, *c_sdaugm* , or dropped with *c_sddrop* .

return value : 0 if successful, -1 or -2 if failed

3.9 File storage and retrieval of individual data Objects.

These routines enable the Application to produce and read back binary files containing the data elements of an Object, plus the information (number of elements, type) required to make the Object usable by the Xdataviewer. An obvious application is when the Object is produced on another computer than the one running the Xdataviewer.
An acceptable Object is an array of one of the following types : int, long, short, float, double, or char[16]. The file where the Object is stored consists of a 4-long word header (containing the number of elements of the array, the size in bytes of each element, the code of the array type, and a spare word), followed by the binary representation of the array elements.

---

int **dv_file_object** (char *objectname, char *filename, char *q)

- **objectname** is the name of an object contained in the current data area space pointed by **q**.
- **filename** is the name of the file where the object has to be stored

The routine produces a binary file, in the format specified above, containing a header and the array associated with the object. This routine is used when the object is contained in a MOPS formatted data area space.

return value : 0 (OK) , -1 (problem with file) or DV_NO_SUCH_OBJECT

---

int **dv_file_array** (char *buf, long nelems, long type, char *filename)

- **buf** is a pointer to the array containing the object data
- **nelems** is the number of elements of the array which have to be saved
- **type** is one of DV_INT, DV_LONG, DV_SHORT, DV_FLOAT, DV_DOUBLE, DV_TEXT (=char[16]). It specifies the data type of the array.
- **filename** is the name of the file where the object has to be stored

The difference with the previous routine is that in this case the data to be saved are not contained in a MOPS object, but rather in a normal array . The type of the array should be one of the above mentioned data types.

return value : 0 (OK) , -1 (problem with file)

---

int **dv_read_object** (char *buf, long maxelem, char *filename)

- **buf** is a pointer to the array which has to receive the object data. The array must have been allocated by the Application. The data type of the array must match the data type of the object saved in the file.
- **maxelem** is the max. number of array elements to be retrieved from the file. If *maxelem* <= 0, the entire file will be read.
- **filename** is the name of the file where the object is stored.

The routine reads back from a file the data array associated with an object. *buf* can point to an array defined by the Application, or could be the pointer returned by *c_sdptr* if the object has to be rewritten into a MOPS.

return value : number of elements read, or -1 (problem with file)

---

int **dv_seek_object** (char *filename, long *nelems, long *elemsz, long *elecod)

- **filename** is the name of the file where the object is stored.
- **nelems** (returned) is the number of elements of the array stored in the file.
- **elemsz** (returned) is the size in bytes of one element
- **elecod** (returned) is the data type of the array : one of DV_INT, DV_LONG, DV_SHORT, DV_FLOAT, DV_DOUBLE, DV_TEXT or DV_INVALID_TYPE

This routine looks at the header of a file supposed to contain an object, and returns information about the contents of the file.

return value : 0 (OK) , -1 (problem with file)

3.10 Query Functions

The values of about all the properties which the Application can set using the routines explained in the previous sections can also be retrieved using the “Query” routines, described in this section.

GENERAL CONFIGURATION QUERIES

---

int **dv_qviews** (int *no_views, char **viewnames[], char *q)

- **no_views** (returned) : the number of views defined in the data space
- **viewnames** (returned) : contains the names of the views.

The routine returns a list of view names. Notice that the space for the list is allocated by the routine, so the Application must later release the space (e.g. *free(viewnames)*).

return value : DV_OK

---

int **dv_qgraphs** (int *no_graphs, char **graphnames[], char *q)

- **no_graph** (returned) : the number of graphs defined in the data space
- **graphnames** (returned) : contains the names of the graphs.

The routine returns a list of graph names. Notice that the space for the list is allocated by the routine, so the Application must later release the space (e.g. *free(graphnames)*).

return value : DV_OK

---

int **dv_qplots** (int *no_plots, char **plotnames[], char *q)

- **no_plot** (returned) : the number of plots defined in the data space
- **plotnames** (returned) : contains the names of the plots.

The routine returns a list of plot names. Notice that the space for the list is allocated by the routine, so the Application must later release the space (e.g. *free(plotnames)*).

return value : DV_OK

VIEW SPECIFIC QUERIES

In all the routines of this class, **viewname** is the name of the view about which information are asked.

---

int **dv_qvwexist** (char *viewname, int *exists, char *q)

- **exists** (returned) : 1 if *viewname* exists, 0 otherwise.

Checks for the existence of a given view.

return value : DV_OK

---

int **dv_qvwcontents** (char *viewname, int *no_graphs, char **graphnames[], int *rowpos[], int *colpos[], int *rowsize[], int *colsize[], char *q)

- **no_graphs** (returned) : the number of graphs attached to the view.
- **graphnames** (returned) : will contain the names of the graphs

- **rowpos** (returned) : will contain the start row of each graph
- **colpos** (returned) : will contain the start column of each graph
- **rowsiz** (returned) : will contain the size in rows of each graph
- **colsiz** (returned) : will contain the size in columns of each graph

Returns information about all the graphs attached to *viewname*. Notice that the space to keep *graphnames, rowpos, colpos, rowsiz, colsiz* is allocated by the routine, and must be later released by the Application (e.g. *free(graphnames); free(rowpos);* etc.) .

return value : DV_OK or DV_NO_SUCH_VIEW

---

int **dv_qvwcreate** (char *viewname, char *viewtitle, char *menuentry, int *no_rows, int *no_cols, char *q)

- **viewtitle** (returned) will contain the title of the view. The programmer should pass the pointer to a character string of (at least) 80 characters.
- **menuentry** (returned) will contain the string that appears in the View List. Same space requirement as above.
- **no_rows** (returned) : number of rows for layout of graphs
- **no_cols** (returned) : number of columns for layout of graphs

Returns the parameters specified in *dv_vwcreate* (and possibly modified by *dv_vwchnames* or *dv_vwchtitle*).

return value : DV_OK or DV_NO_SUCH_VIEW

GRAPH SPECIFIC QUERIES

In all the routines of this class, **graphname** is the name of the graph about which information are asked.

---

int **dv_qgrexist** (char *graphname, int *exists, char *q)

- **exists** (returned) : 1 if *graphname* exists, 0 otherwise.

Checks for the existence of a given graph.

return value : DV_OK

---

int **dv_qgrattachments** (char *graphname, int *no_views, char **viewnames[], int *rowpos[], int *colpos[], int *rowsize[], int *colsize[], char *q)

- **no_views** (returned) : the number of views to which the graph is attached.
- **viewnames** (returned) : will contain the names of the views
- **rowpos** (returned) : will contain the start row of the graph in each view
- **colpos** (returned) : will contain the start column of the graph in each view
- **rowsiz** (returned) : will contain the size in rows of the graph in each view
- **colsiz** (returned) : will contain the size in columns of the graph in each view

Returns information about all the views to which *graphname* is attached. Notice that the space to keep *viewnames, rowpos, colpos, rowsiz, colsiz* is allocated by the routine, and must be later released by the Application (e.g. *free(viewnames); free(rowpos);* etc.) .

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrcontents** (char *graphname, int *no_plots, char **plotnames[], char *q)

- **no_plots** (returned) : the number of plots attached to the graph.

- **plotnames** (returned) : will contain the names of the plots

Returns information about all the plots attached to *graphname*. Notice that the space to keep *plotnames* is allocated by the routine, and must be later released by the Application (e.g. *free(plotnames);)*

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrcreate** (char *graphname, char *title, char *xlabel, char *ylabel, char *q)

- **title** (returned) will contain the title of the graph. The programmer should pass the pointer to a character string of (at least) 80 characters.
- **xlabel** (returned) will contain the x-axis label for the graph. Same space requirement as above.
- **ylabel** (returned) will contain the y-axis label for the graph. Same space requirement as above.

Returns the parameters specified in *dv_grcreate* (and possibly modified by *dv_grchnames*).

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qglabels** (char *graphname, int *nlabels, char **labels[], char *q)

- **nlabels** (returned) : the number of labels defined for the graph.
- **labels** (returned) : will contain a list of label strings

The labels associated with the graph *graphname* are returned. Notice that the space to keep *labels* is allocated by the routine, and must be later released by the Application (e.g. *free(labels);)*

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrlimits** (char *graphname, double *xmin, double *xmax, double *ymin, double *ymax, char *q)

- **xmin** (returned) is the minimum x coordinate
- **xmax** (returned) is the maximum x coordinate
- **ymin** (returned) is the minimum y coordinate
- **ymax** (returned) is the maximum y coordinate

The x and y limits associated with *graphname* are returned.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrpercent** (char *graphname, double *xpercent, double *ypercent, char *q)

- **xpercent** (returned) is the margin around the x data (as a percentage of the whole x range)
- **ypercent** (returned) is the margin around the y data (as a percentage of the whole y range)

Returns the two percentages.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrscales** (char *graphname, int *xscale, int *yscale, char *q)

- **xscale** (returned) is the type of scale used for the x-axis (one of DV_LINEAR, DV_LOG or DV_TIME)
- **yscale** (returned) is the type of scale used for the y-axis (one of DV_LINEAR, DV_LOG)

Returns the type of scales associated with the graph.

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrlist** (char *graphname, int *listopt, int *mergeflag, char *q)

- **- listopt** (returned) is one of DV_NOLIST, DV_LIST
- **- mergeflag** (returned) is one of DV_NOMERGE, DV_MERGE

Returns the values of *listopt* and *mergeflag*, defining the way in which the graph will be represented (graphic or text mode, merging or not of plots with different x coordinates).

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrliststyle** (char *graphname, int *liststyle, char *q)

- **- liststyle** (returned) is one of DV_NOLISTFULL, DV_LISTSHORT

Returns the values of *liststyle*, defining the way in which the graph will be saved onto a file in ASCII format (printing also values of the auxiliary objects, or only the plots data objects).

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrnpoints** (char *graphname, int *npoints, char *q)

- **- npoints** (returned) is the limit on the number of points which can be displayed on the graph (-1 meaning “no limit”).

Returns the limit possibly set by *dv_grnpoints* .

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrsetgrid** (char *graphname, int *gridstyle, char *q)

- **- gridstyle** (returned) is the default grid setting for the graph (one of DV_GRID_OFF, DV_GRID_ON, DV_GRID_FULL)

return value : DV_OK or DV_NO_SUCH_GRAPH

---

int **dv_qgrsetzeroline**(char *graphname, int *zeroline, char *q)

- **- zeroline** (returned) is the default zeroline setting for the graph (one of DV_ZEROLINE_OFF, DV_ZEROLINE_ON, DV_ZEROLINE_XY)

return value : DV_OK or DV_NO_SUCH_GRAPH

PLOT SPECIFIC QUERIES

In all the routines of this class, **plotname** is the name of the plot about which information are asked.

---

int **dv_qplexist** (char *plotname, int *exists, char *q)

- **- exists** (returned) : 1 if *plotname* exists, 0 otherwise.

Checks for the existence of a given plot.

return value : DV_OK

---

int **dv_qplattachments** (char *plotname, int *no_graphs, char **graphnames[], char *q)

- **- no_graphs** (returned) : the number of graphs to which the plot is attached.
- **- graphnames** (returned) : will contain the names of the graphs

Returns the list of the graphs to which *plotname* is attached. Notice that the space to keep *graphnames* is allocated by the routine, and must be later released by the Application (e.g. *free(graphnames);)*

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplcreate** (char *plotname, int *plottype, char *xdata, char *ydata, char *q)

- **plottype** (returned) will contain the type of the plot.
- **xdata** (returned) will contain the name of the MOPS object containing the x data for the plot. The programmer should pass the pointer to a character string of (at least) 80 characters.
- **ydata** (returned) will contain the name of the MOPS object containing the y data for the plot. Same space requirement as above.

Returns the parameters specified at plot creation. If the plot is 1-dim, *xdata* will be an empty string.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qpldtform** (char *plotname, int *dataform, char *q)

- **dataform** (returned) is the form of the data for the plot (DV_RANDOM, DV_XMONOTONIC, or DV_XUNIFORM).

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplreadwrite** (char *plotname, int *readwrite, char *q)

- **readwrite** (returned) is one of DV_READ (readonly plot) , DV_WRITE (editable plot)

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplhis** (char *plotname, int *histcol, char *q)

- **histcol** (returned) is the histogram colour associated with the plot.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qpllin** (char *plotname, int *linecol, char *q)

- **linecol** (returned) is the line colour associated with the plot.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplmar** (char *plotname, int *markercol, int *markertype, char *q)

- **markercol** (returned) is the marker colour associated with the plot.
- **markertype** (returned) is the marker type associated with the plot.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplcol** (char *plotname, char *colours, char *q)

- **colours** (returned) is the name of the MOPS colour object associated with the plot. The programmer should pass the pointer to a character string of (at least) 80 characters.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplcolouredline** (char *plotname, int *value, char *q)

- **value** (returned) is a 0 / 1 flag reflecting the value set by *dv_plcolouredline* (or the default)

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplcoltxt** (char *plotname, char *colours, char *q)

- **colours** (returned) is the name of the MOPS colour object associated with the text representation of the plot. The programmer should pass the pointer to a character string of (at least) 80 characters.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qpltxt** (char *plotname, char *text, char *q)

- **text** (returned) is the name of the MOPS text object containing the labels for the different data points of the plot. The programmer should pass the pointer to a character string of (at least) 80 characters.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplerrorbars** (char *plotname, char *x_left, char *x_right, char *y_top, char *y_bottom, char *q)

- **x_left** (returned) is the name of the MOPS object containing the error values for the horizontal left direction. The programmer should pass the pointer to a character string of (at least) 80 characters.
- **x_right** (returned) is the name of the MOPS object containing the error values for the horizontal right direction. Same space requirement as above.
- **y_top** (returned) is the name of the MOPS object containing the error values for the vertical upwards direction. Same space requirement as above.
- **y_bottom** (returned) is the name of the MOPS object containing the error values for the vertical downwards direction. Same space requirement as above.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qpllist** (char *plotname, int *listopt, char *q)

- **listopt** (returned) is a flag (DV_LIST or DV_NOLIST). Its value defines whether the plot will be initially displayed in text or graphical format.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qpllistform** (char *plotname, char *format_x, char *format_y, char *q)

- **format_x** (returned) contains the format used to list the x values for the plot. The programmer shoud pass a pointer to a character string of (at least) 8 characters.
- **format_y** (returned) contains the format used to list the y values for the plot. The programmer shoud pass a pointer to a character string of (at least) 8 characters.

return value : DV_OK or DV_NO_SUCH_PLOT

---

int **dv_qplnpoints** (char *plotname, int *npoints, int *first, char *q)

- **npoints** (returned) is the maximum number of points to be displayed for the plot. If *npoints* = 0 all the points will be displayed.
- **first** (returned) is the index of the first point to be displayed.

return value : DV_OK or DV_NO_SUCH_PLOT

MISCELLANEOUS QUERIES

---

int **dv_qobexist** (char *obname, int *exists, int *type, char *q)

- **obname** is the name of the MOPS object whose existence is queried
- **exists** (returned) is 1 if the object exists, 0 otherwise.
- **type** (returned) will be one of DV_INT, DV_FLOAT, DV_LONG, DV_DOUBLE, DV_SHORT, DV_TEXT or DV_INVALID_TYPE

The routine looks for a MOPS object *obname* in the data space pointed by *q*. If it finds it, it sets *type* to the type of the object. Only the above mentioned types are acceptable by the Xdataviewer.

return value : DV_OK

---

int **dv_qversion** (int *version , char *q)

- **version** (returned) is the Xdataviewer version (currently 6) used to configure the data space pointed by *q*.

return value : DV_OK, DV_NO_SUCH_OBJECT, DV_NO_SUCH_VIEW

3.11 Embedded Dataviewer Specific Routines

A small number of routines have been written, to deal with the peculiarity of having the functionality of the Xdataviewer embedded in the Application. Some routines are used to simulate the passing of command line arguments (which data to be loaded, which options to be used). Other ones (**dv__kick**, **dv__kick_and_save**, already described) implement the Application request to update the screen. another class can be used to handle common information between the Application part of the process and the Xdataviewer part.
The particularity of all the routines specific to the Embedded Dataviewer is that their name starts by "dv__" (dv, followed by a double underscore character). The exceptions to this rule are routine **dv_before_main**(), to be called during the initialization of the Application, and **dataviewer_initialize**(), which can be used when the programmer wants to pass a new data space pointer to the Embedded Dataviewer.
A more detailed description on how to set up an Application using the Embedded Dataviewer, and when to use the following routines, will be found in chapter "The Embedded Dataviewer".

ROUTINES TO SPECIFY DATA TO BE LOADED

---

int **dv__sharename** (char *mopsname)

- **mopsname** is the name of the mops to be loaded.

Makes the Embedded Dataviewer loading the specified mops. According to the MOPS naming convention, if the first character is @, the rest of the name will be used as an absolute path to the shared memory key identification file, otherwise the name will be added the prefix $HOME/shardat .

return value : 0

---

int **dv__pointer** (char *ptr)

- **ptr** points to a memory buffer, defined by the Application, used as a storage for the data space.

The Application and the Embedded Dataviewer, being part of the same process, share the same addressing space. Therefore the Xdataviewer data, still formatted using the MOPS functions, can be contained in a memory buffer private to the process. *ptr* is the pointer to that memory buffer.

return value : 0

---

int **dv__filename** (char *filename)

- **filename** is the name of a file containing the Xdataviewer data (in MOPS format, previously saved by an Application or by the Xdataviewer itself). If *filename* starts with two "slash" characters ("//"), it will be connsidered as a URL and searched for on the World Wide Web.

The file will be reloaded in a newly allocated memory buffer, which will become the data space.

return vaue : 0 or -1 (problems reloading the file)

---

int **dv__sequencename** (char *sequencename)

- **sequencename** is the name of the sequence to be loaded

A sequence is a file containing the names of data files readable by the Xdataviewer. When a sequence is loaded, the Xdataviewer user has the possibility of navigating between these different data files. This mode can be very useful for presenting the kind of results which can be obtained via the Application. When *dv__sequencename* is called, the first file in the sequence will be automatically loaded.

return value : 0 or -1 (problems with the sequence file or with the first datafile in the sequence)

---

int **dataviewer_initialize**()

Routine to be called every time the programmer has selected a new data space using one of the previous routines.

return value : 0

---

int **dv__clearchangeptr** ()

This routine should be called **before** *dv__pointer* when the Application is already using a data space contained in an internal memory buffer, and it wants to quit temporarily the first buffer and use a second one. The routine avoids memory leakage.

return value : 0

---

int **dv__freeptr** (char *ptr)

- **ptr** is a data space pointer previously allocate via malloc and declared by *dv__pointer*(ptr) .

This routine should be used to clean the memory when access to the data space pointed by *ptr* is not needed anymore. The routine avoids memory leakage.

return value : 0

OPTION SPECIFICATION FOR THE EMBEDDED DATAVIEWER

---

int **dv__initialview** (char *viewname)

- **viewname** is the name of the view to be initially displayed

Sets the name of the view to be initially displayed after the creation of the Embedded Dataviewer.

return value : 0

---

int **dv__initialgraph** (char *graphname)

- **graphname** is the name of the graph to be initially displayed

Sets the name of the graph to be initially displayed after the creation of the Embedded Dataviewer.

return value : 0

---

int **dv__argument** (char * optionname, char *optionvalue)

- **optionname** is the name corresponding to the option to be set
- **optionvalue** contains the value (if any) to be assigned to the option

This is a generic routine to be called before the Embedded Dataviewer creation, to simulate the passing of command line arguments. We list some of the possible options in the format **OPTION** <**value**>

empty , to specify that no initial data will be provided.

WHITE (or **white**) , to start the Embedded Dataviewer with a white background.

BLUE (or **blue**) , to start the Embedded Dataviewer with a blue background.

GREY (or **grey**) , to start the Embedded Dataviewer with a grey background.

background <**colour**> , to start the Embedded Dataviewer with a <colour> background

useblack : kludge to make DV_BLACK points visible and DV_WHITE ones invisible.

W <**xpos_ypos_xsize_ysize**> , to specify size and position for the Embedded Dataviewer window

DOT <**dot_size**>, **BOX** <**box_size**>, **ARC** <**arc_size**>, **CIRCLE** <**circle_size**> , to define the size, in pixels, of the markers DV_USER_DOT, DV_USER_BOX, DV_USER_ARC, DV_USER_CIRCLE.

mole , to use larger fonts when listing plots and graphs.

home , to use the value of the HOME variable to initialize file selection boxes.

fixedsource, to suppress the possibility of changing data area space from the Xdataviewer..

nolistplot or **nolistgraph**, to suppress the possibility of listing individual plots or complete graphs.

colorprinter <**printername**> : to set the default colour printer

printer <**printername**> : to set the default bw printer

textprinter <**printername**> : to set the default text printer

helpdv <**helpfile**> : to change the Help file for the Embedded Dataviewer part

helpfile <**helpfile**> : to set the Help file for the Application part

return value : 0 or -1 (illegal option)

---

int **dv__gimme_window**()

returns the Window Id of the Window containing the Embedded Dataviewer part.

COMMUNICATION BETWEEN THE APPLICATION AND THE EMBEDDED DATAVIEWER

---

int **dv__kick**() : discussed in section 3.2
int **dv__kick_and_save**() : discussed in section 3.2
int **dv__mode_set** (int option, int value) : discussed in section 3.1

3.12 Miscellaneous

DEBUGGING

---

void **dv_error** (int error)

- **error** is an error code returned by one of the Callable Interface routines

Prints an error message associated with the given error.

SAVING AND RESTORING XDATAVIEWER CONFIGURATION
These routines are not really up to date, as they are of little practical use.

---

int **dv_confsave** (filename, q)

- **filename** is the file into which the configuration has to be saved

Saves the current Xdataviewer configuration (views, graphs, plots) into the named file, in text format.

return value : DV_OK, or DV_CANNOT_OPEN_FILE

---

int **dv_confrestore** (filename, q)

- **filename** is the file from where the configuration has to be restored

Restores the Xdataviewer configuration from the named file (in the format produced by *dv_confsave*). This routine will replace any existing views, graphs and plots.

return value : DV_OK, or DV_CANNOT_OPEN_FILE (or other kind of errors, if the file has been modified, or if the data space is too small to contain all the views, graphs, plots defined in the file)

3.13 Obsolete (old fashion MOPS hiding routines)

Inspite of the HP_UX Dataviewer being completely compatible with the previous Apollo version, the Application Programmer should better avoid using the functions listed herebelow. These functions, although still supported, are not completely safe, and may also lead to a bad programming style. Moreover, most of these functions are very easily replaceable by functions within the Mops library, which should be used instead.

- **dv_dsattach** : use c_sdacc instead .
- **dv_dserase** : use c_sdkill instead .
- **dv_dsinit** : use (c_sdkill), c_sdalloc , c_sdini , dv_init instead .
- **dv_dsrelease** : use c_sdquit instead .
- **dv_obsize** : not needed if you do not use dv_dsinit .
- **dv_dsdisplay** : it hides the forking of the Dataviewer as a separate process. It is better either to explicitly start the Dataviewer as a separate process (from another window, or from the Console Manager) , or to use the Embedded Version described later.
- **dv_obcreate** : use c_sdbook or c_sdzero, and possibly c_sdwrit (or work with pointer, c_sdptr)
- **dv_obread** : use c_sdfind, c_sdreac instead (or better work with the object pointer, c_sdptr)
- **dv_obreplace** : use c_sdwrit instead (or work with the object pointer , c_sdptr)

---

int **dv_dsattach** (char *data_area_name, int mode, char **q)

- **- data_area_name** is the name of the MOPS to be attached
- **- mode** is DV_READ or DV_WRITE
- **- q** is the returned pointer to the MOPS

(Re)Attaches the MOPS. **Use c_sdacc instead** !

return value : DV_OK, DV_SDACC_ERROR, DV_INVALID_MODE

---

int **dv_dserase** (char *data_area_name)

- **- data_area_name** is the name of the MOPS to be deleted from the system

Deletes the MOPS *data_area_name* from the computer. The MOPS (Unix shared memory) normally survives until explicitly deleted, or until the computer is rebooted. **Use c_sdkill instead !**

return value : DV_OK, DV_SDKILL_ERROR

---

int **dv_dsrelease** (char *q)

- **- q** is the MOPS pointer returned by *c_sdacc*, or *c_sdalloc*, or *dv_dsattach*

Gives up access to the MOPS (useful if the Xdataviewer has to get access to the MOPS, for instance to read the data). After this call *q* becomes meaningless. The MOPS has to be then reattached (*c_sdacc* or *dv_dsattach*) . **Use c_sdquit instead !**
Notice that *c_sdquit* is automatically called from inside the *dv_kick* routine, when the Application wants to tell the Xdataviewer that new data is ready. So typically the only case in which you explicitly may want to release the MOPS is before starting the Xdataviewer, and you can also avoid this if you access

the MOPS with the SD_RTC flag (*c_sdacc*).

return value : DV_OK, DV_SDQUIT_ERROR

---

int **dv_obsize** (int size)

- **- size** is the “default object size”

(**NOT !**) to be used together with *dv_dsinit*, which is dangerous and conceptually buggy. You do not need it if you do not use *dv_dsinit*. For a proper usage of MOPS, you should be able to estimate with precision how much shared memory you need, on the bases of the data object you want to define and of your Xdataviewer configuration (see *dv_space*). We do not take any responsibility for errors generated by usage of this and the next routine !

---

int **dv_dsinit** (char *data_area_name, int no_views, int no_graphs, int no_plots, int no_objects, char **q)

- **- data_area_name** is the name of the MOPS to be created
- **- no_views** is the maximum number of views foreseen by the programmer.
- **- no_graphs** is the maximum number of graphs foreseen by the programmer.
- **- no_plots** is the maximum number of plots foreseen by the programmer.
- **- no_objects** is the maximum number of mops objects foreseen by the programmer.
- **- q** is the returned pointer to the MOPS

DO NOT USE THIS ROUTINE ! Use (c_sdkill), c_sdalloc , c_sdini , dv_init instead .
Goal of *dv_dsinit* is to re-create (use *c_sdalloc* instead) a MOPS data_area_name (deleting it if it already existed, use *c_sdkill*), and to configure it to accommodate the specified number of objects (use *c_sdini*) and of Xdataviewer views, graphs and plots (use *dv_init*). The main problem is that the size of the MOPS will not be computed according to what it should contain, but on the basis of a guess depending on *no_objects* and on the default “object size” (512 bytes, but can be modified by *dv_obsize*).

return value : DV_OK, DV_NO_ROOM, DV_CANNOT_CREATE_FILE, DV_SDKILL_ERROR, DV_SHM_ALLOCATION_ERROR, DV_NO_SHARDAT or DV_SDINI_ERROR

---

int **dv_dsdisplay** (char *viewname, char *q)

- **- viewname** is the name of the view to be originally displayed
- **- q** is the pointer to the mops

DO NOT USE THIS ROUTINE ! This routine is supposed to start the Xdataviewer from inside the current process (it uses the “system” call to launch it). It does not do it in a proper way, and it contains a bug (wrong assumption on finding the MOPS name inside the MOPS). It should be avoided. There are better ways for starting the Xdataviewer, both from inside the process and from the Console Manager. They will be explained in detail in the next chapter and in the examples of chapter 9.

4

How To Start The Xdataviewer

This chapter contains a description of the different command line arguments recognized by the Xdataviewer, as well as guidelines explaining convenient ways for starting the Xdataviewer in different situations.

4.1 The Xdataviewer Command Line Arguments

The Xdataviewer may be started in one of the following ways :

- Xdataviewer <**mopsname**> [initialview] [initialgraph] [[options]] (*to start by using a MOPS*)
- Xdataviewer <***filename***> [initialview] [initialgraph] [[options]] (*to start by using a file*)
- Xdataviewer -**sequence** <sequencename> [[options]] (*to start by loading a sequence of files*)
- Xdataviewer -**empty** [[options]] (*to start without any data attached*)

Notice that if filename starts by //, it will be considered as a URL, and searched for on the World Wide Web.
Ex. Xdataviewer *//www.cern.ch/myjunk_URL

Every option is composed by a keyword prefixed by the "-" character (ex. "-*white*"), and is followed by the arguments needed by the option itself. **initialview** (**initialgraph)** is the name of a view (graph) to be shown when the dataviewer is started (optional).

The following options are recognized ;

OPTIONS TO CONTROL THE BACKGROUND COLOR

- **-** -**WHITE** (or -**white**) : starts the Xdataviewer with a white background (instead than the default black one)
- **-** -**GREY** (or -**grey**) : uses a grey background
- **-** -**BLUE** (or -**blue**) : uses a blue background
- **-** -**background** <**colour**> (Xwindow option) to set the background to a given colour
- **-** -**useblack** : this option is used to reverse the normal behaviour of the Xdataviewer, which draws points whose color is DV_BLACK using the background colour (making them always invisible), and points whose color is DV_WHITE using the opposite colour (making them always visible). If the option -*useblack* is used, points in DV_BLACK will be always visible, and points in DV_WHITE invisible.

OPTIONS TO CONTROL THE DATAVIEWER WINDOW SIZE

- **-** -**X** <**xsize**> : to specify the horizontal size of the Xdataviewer Graphic Window (miminum value is 584, default is 800).
- **-** -**Y** <**ysize**> : to specify the vertical size of the Xdataviewer Graphic Window (miminum value is 200, default is 500).
- **-** -**W** <**xpos_ypos_xsize_ysize**> : to specify size and position of the Xdataviewer Window.

NOTE : in order not to be trapped behind the CERN/SL Console Manager, the X argument "-geometry +0+64" should be used.

OPTIONS TO CONTROL THE DATAVIEWER OUTPUT DESTINATION (SEE ALSO PRINTER SETTING DIALOG)

- **-** -**colorprinter** <**colorprintername**> . To define the name of the Colour Printer to be used for colour screen dumps. The default is *pcrtxp* (CERN Control Room Colour Printer).

- **--printer** <**printername**> . To define the name of the BW printer to be used for black and white screen dumps. The default is the default *lp* printer for the hp_ux where the Xdataviewer is running.
- **--textprinter** <**printername**> . To define the name of the printer for alphanumerical output.

NOTE : an entire new popup page has been added to the Xdataviewer, to provide more flexibility in printer selection and in .ps or .eps file output.

OPTIONS TO CONTROL THE USER DEFINED MARKERS

- **--DOT** <**size**> . To define the size of the DV_USER_DOT Marker Type. The size of the squared dot will be **2*size + 1**.
- **--BOX** <**size**> . To define the size of the DV_USER_BOX Marker Type. The size of the squared box will be **2*size + 1**.
- **--ARC** <**size**> . To define the size of the DV_USER_ARC Marker Type. The ray of the empty circle will be **size**.
- **--CIRCLE** <**size**> . To define the size of the DV_USER_CIRCLE Marker Type. The ray of the circle will be **size**.

OPTIONS TO CONFIGURE THE INTERFACE

- **--nolistplot** : To disable the possibility of listing single plots
- **--nolistgraph** : To disable the possibility of listing entire graphs
- **--fixedsource** : makes the LOAD/SAVE Pulldown Menu inactive, removing the possibility of changing the Xdataviewer input source from the Xdataviewer itself.

OPTION TO CHANGE THE DEFAULT DIRECTORIES FOR FILE SELECTION BOXES

- **--home** : if this option is used, the default directory for the mops selection box will be **$HOME/shardat** (instead than "**/usr/tmp**"), and the one for file selection will be **$HOME** (instead than "**.**").

MISCELLANEOUS OPTIONS

- **--wait** <**delay**> : The Xdataviewer waits up to <delay> seconds if the Mops does not yet exist.
- **--helpfile** <**file**> : To tell the Xdataviewer the name of the Application Help File (See section on Help Facility)
- **--mole** : To use bigger fonts when listing plots and graphs in ASCII mode.

4.2 Starting the Xdataviewer together with the Application

4.2.1 Starting under the control of a common parent process (e.g. Console Manager)

We want to start the Application and the Xdataviewer more or less at the same time, under the control of a common parent process. The problem is how to make sure that the Application will have already created the MOPS (or the file) required by the Xdataviewer when this last process will try to access it.
One possibility is to use the **-wait** option described in the previous section. In this case the Xdataviewer will be ready to wait for a reasonable delay before giving up the possibility of attaching the MOPS.
A second possibility is to use a trigger file. In this case, the Application will create a dummy file when the MOPS is ready, and the Xdataviewer command will be contained in a script, preceded by a process that has to wait for the dummy file before completing. For example, the script could contain the two following lines :

```
/user/bim/tools/wait_file /tmp/tune1000
/user/bim/DATAVIEWER/Xdataviewer @/usr/tmp/tune1000_mops -W 20_444_1200_444 -name bom_tune1000
```

where **/tmp/tune1000** is the dummy file created by the Application, and **wait_file** is a simple program that waits for a file and deletes it.
This method is not absolutely sure (for instance, the dummy file might already exist before the Application is run), but it is simple and usually works.

4.3 Starting the Xdataviewer directly from the Application process

By doing this, the issue addressed in the previous section is automatically solved. The problem, in this case, is how to achieve a good system independency between the Application process and the Xdataviewer process. The way we recommend is the following :

- write a script containing the command line needed to start the Xdataviewer **as a background process** (use the **&** character) .
- execute the script from the program via the “system” system call.

Ex.
in Script File “/user/junk/mystart” :
Xdataviewer @/usr/tmp/junkmops view1 [options] -geometry +0+64 &

- inside the Application program insert a system call to start the script.

Ex.
* start dataviewer */
system(“/user/junk/mystart”)

- Do not forget to make /user/junk/mystart executable by “*chmod +x*”

This way is better than the often used f*ork-exec* or *dv_dsdisplay* approach, because the Xdataviewer is then running independently from the Application Program, instead of becoming a child of it. If the Application Program needs it, it can retrieve the process identifier of the Dataviewer using the function **dv_get_dv_processid** .

A similar behaviour can anyway be achieved by using *fork-exec* and executing the Xdataviewer in background mode (using the **&** character).

5

The Xdataviewer Graphical User Interface

This chapter explain in details the different parts of the Xdataviewer Graphical Interface (GUI) and how the User can interact with the data.

5.1 The basic parts of the Xdataviewer GUI

In Fig. 5.1 a typical Xdataviewer window is shown. The three top lines of the window constitute the "***control part***" of the interface, and contain several "control buttons" via which the User can select the data to be shown, decide the operations to be performed on the data (listing, editing, zooming, selecting application dependent options, etc). The bottom part ("***graphical area***") is dedicated to the data representation. In this part, graphs and plots (and possibly pictures) are represented in the form chosen by the Application and eventually modified by the User.

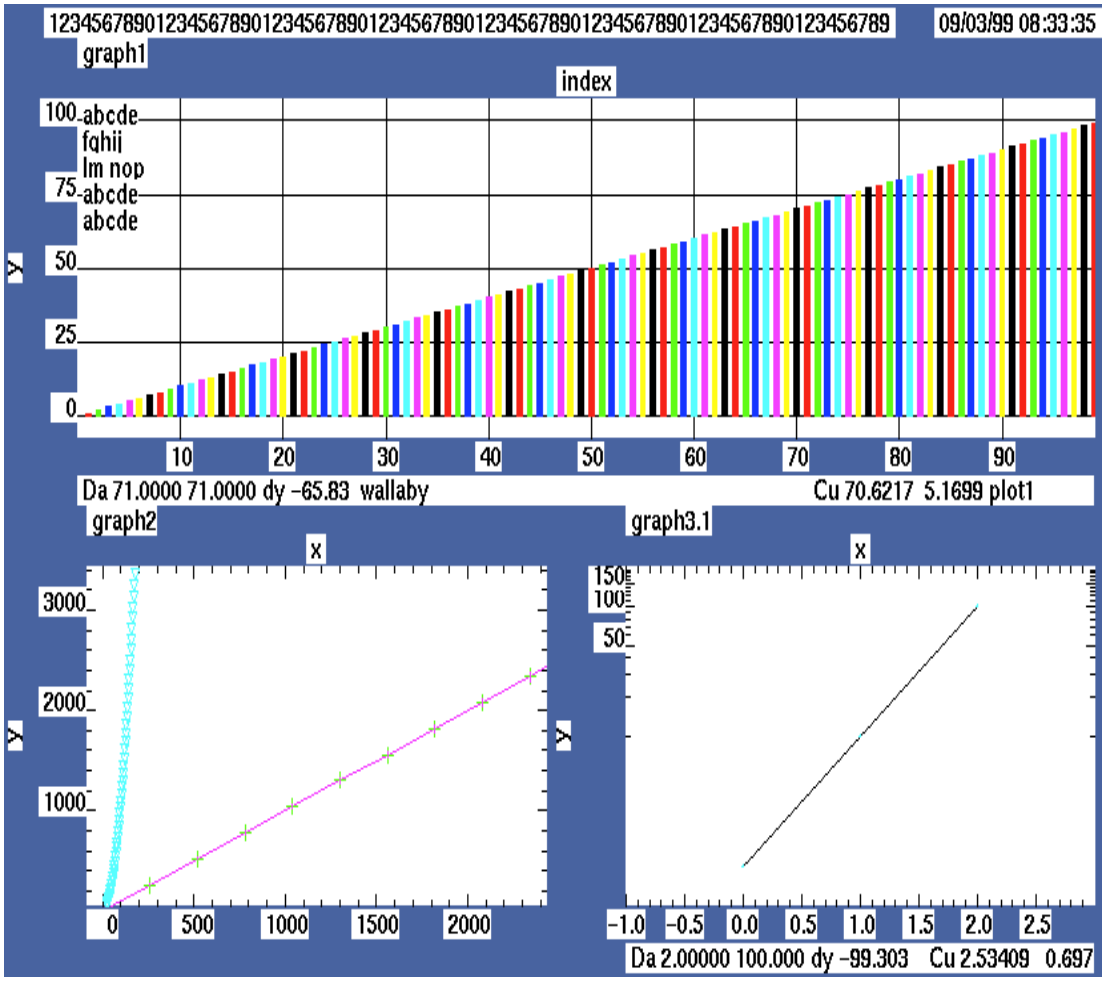

Figure 5.1. A typical Xdataviewer window. A view with three graphs is displayed.

In Fig. 5.1 a view with 3 graph is being displayed. The top graph contains one plot of type "histogram", to which an auxiliary colour object has been associated, to define a different colour for each point. Graph 2 contains two different plots, one of type "marker", whose points, marked as small triangles, sit on a parabolic function. The other plot, of type "line", has data points marked by "plus" markers and sitting on a straight line. Vertical errorbars have been associated with the data points of this second plot. The third graph (graph3.1) has a vertical logarithmic scale. Two different types of grids have been associated with the graphs. Another important feature non visible on the picture is the "active cursor"; every time the mouse cursor enters the region where the data are represented, the position of the cursor is updated on the "***cursor line***" below the graph containing the cursor, together with information relative to the closest data point to the cursor. Graph 2 does not display this information, because it was never entered by the cursor. The short text strings in the top left corner of graph 1 have been set via *dv_grlabels*, while

the text label “wallaby” appearing in the cursor line for the same plot comes from a label text object associated to the plot via *dv_pltxt*.
Individual plots or complete graphs can be also represented in text format, as shown in Fig 5.2.

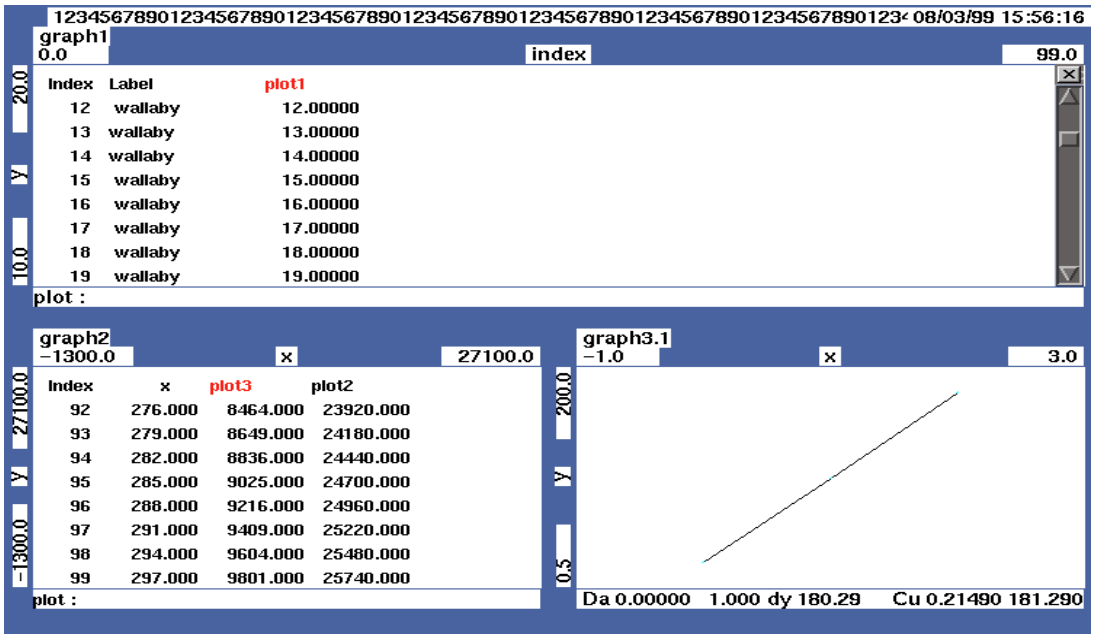

Figure 5.2. The same view as in Fig. 5.1. Two graph are represented in text format.

In this case a *scrollbar* is added to the concerned graphs, to speed up scrolling operations for plots with a large number of points.

5.2 The Control Part

Fig 5.3 shows in more detail the Control Part of the Xdataviewer Interface.

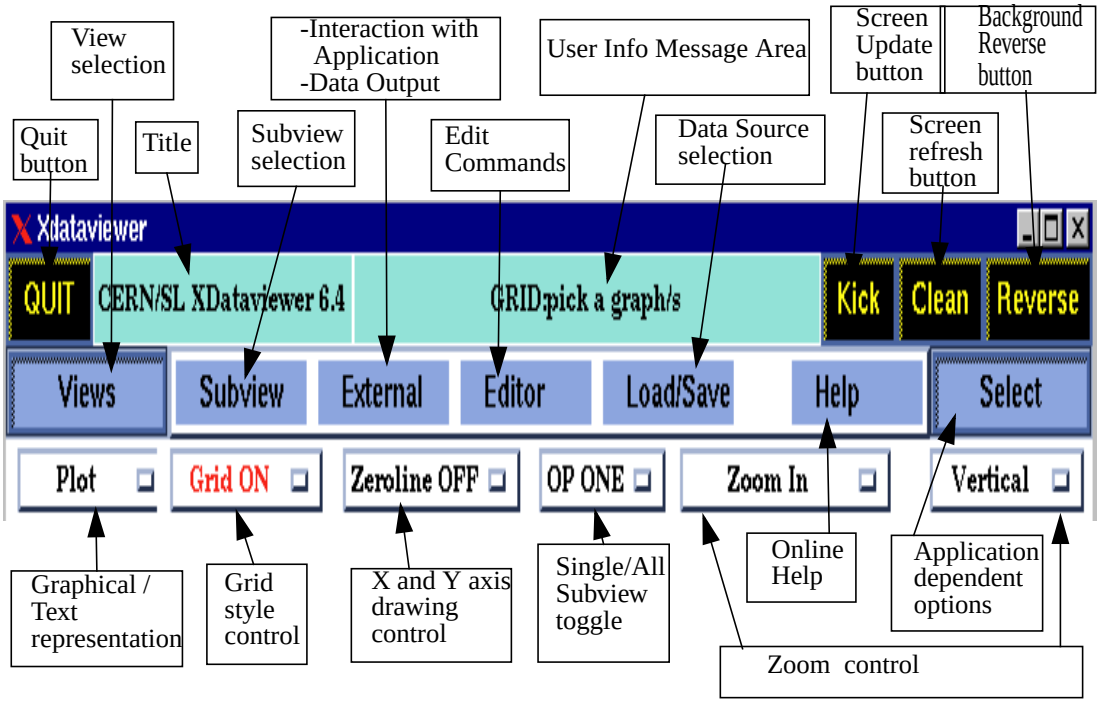

Figure 5.3 . The different components of the Xdataviewer GUI Control Part

The interface has been written using the XWindow C code generator Xcreator [5], and is therefore easily modifiable. Behind the statically visible buttons, many different menus provide the User with all the required functionality. We will describe the different components one by one.

THE FIRST LINE OF THE INTERFACE
This line contains 4 buttons (QUIT, KICK, CLEAN and REVERSE) , plus the Xdataviewer logo and the "User Info message Area".

- the **QUIT** button can be used to terminate the Xdataviewer process.
- the **logo** contains information about the current version.
- the **User Info Message Area** is used by the Xdataviewer to show the User short instruction messages (e.g. "GRID:pick a graph/s") or error conditions (e.g. "Not valid, pick again")
- the **Kick** button will force the Xdataviewer to reread the data space and redisplay the current view. This could be useful if, for any reason, the Application and the Xdataviewer had lost the synchronism.
- the **Clean** button will clean the screen (some times the image of the cursor is left even if the cursor is outside from the graphical part) and redraw the current data. It also removes the "cursor line" from every graph.
- the **Reverse** button switches the background from black to white and viceversa. All points drawn in white over black will be automatically drawn in black over white, and viceversa.

THE SECOND LINE OF THE INTERFACE
This line contains two buttons to popup lists (VIEWS and SELECT), and a menubar containing 5 buttons (SUBVIEW, EXTERNAL, EDITOR, LOAD/SAVE, HELP), each of which will pulldown a menu.

- The **VIEWS** button

  When the User clicks on this button, a popup shell will show a scrolled list containing the names of all the available Views. By clicking on one of these names, the newly selected View will be displayed. The popup shell will disappear automatically when entered and then exited by the mouse cursor.

- The **SUBVIEW** Pulldown Menu

  By choosing one of the options of the menu which appears when SUBVIEW is clicked, the User is able to display a selected subset of the graphs contained in the current View.

  - **ALL** : all graphs will be displayed
  - **ONE** : only one graph will be displayed. The User will select it by moving the cursor to it and clicking.
  - **N** : the User will select the graphs to be displayed by clicking them. To signal the completion of the selection, the User will click twice on the last graph.

- The **EXTERNAL** Pulldown Menu

  This menu, activated when the User clicks on EXTERNAL, contains 5 options dedicated to the communication with the Application, and 5 options dedicated to the output of the data in different ways. The first 5 options can actually have different names, or they could have been disabled, as they are under the control of the Application Programmer through routine *dv_set_external_menu* .

  - **Write Data** : forces a rewriting of the contents of the view currently displayed into the "*data area space*" (let us remind the reader that, when a view is displayed, data is copied from the data area space to a temporarily allocated local working space, and that any modification to the view is not propagated to its image in the data area space, unless explicitly required, for instance by using this option).
  - **Write Data and Signal** : same as before, but it also signals the Application Program about the event (Unix signal SIGUSR1).
  - **Send_Signal** : sends a Unix signal SIGUSR1 to the Application Program.
  - **Stop Incoming signal** : prevents the Application from sending signals to the Xdataviewer
  - **Start Incoming Signal** : reenables the signals coming from the Application.

- **Print Data On Printer** : prints the values of all the data points of all the plots in a graph (which the User will select by clicking it) . The output goes to the *Text Printer*, as specified by the *Printer Setting* configuration shell.

- **Plot Data On Printer** : makes a black and white copy of the graphical area of the Xdataviewer. The output goes to the *Black and White Printer*, or to a file, or to both, as specified by the *Printer Setting* configuration shell.

- **Color Plot On Printer** : makes a colour copy of the graphical area of the Xdataviewer. The output goes to the *Colour Printer*, or to a file, or to both, as specified by the *Printer Setting* configuration shell.

- **Save Graph To File** : writes into a file the values of all the data points of all the plots in a graph (which the User will select by clicking it) . The file selection is helped by a popup shell, in which the User can select a directory and a file name. A default name, self incrementing at each Save Graph operation, is already provided. The User can save a short remark together with the data.

- **Printer Setting** : when this option is selected, a dialog shell will appear (Fig. 5.4). Using the options contained in this shell, the User can independently configure the Output for three different operations : Colour Plots, Black & White Plots, Text Output. To each of these operations it is possible to assign a different printer. It is also possible to address the output to a *.ps* or *.eps* file and to modify the size of the output page. The commands which will be executed when the output operation is actually performed (by using one of the previously described options) are explicitly shown on the screen.

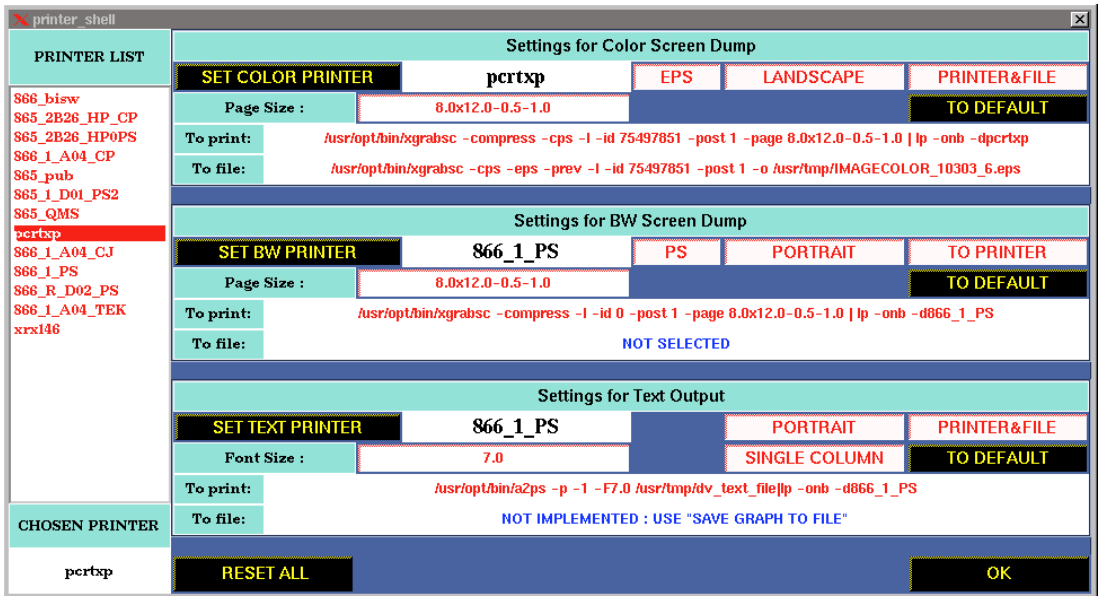

Figure 5.4 . The Xdataviewer Printer Setting Dialog Shell.

- The **EDITOR** Pulldown Menu

The options contained in this menu enable the User to modify the data contents of a plot. To be edited, the plot must have been declared writeable by the Application Programmer. If the editing operation is carried on inside a graph containing more than one editable plot, the plot selected will be the one with a data point “closest” to the cursor. This will depend also on the data format selected for the plots (DV_RANDOM or DV_MONOTONIC). In fact, the editing facility was originally developed to allow modifications of equipment functions during the LEP energy ramp. It is intended to work mainly with monotonic, possibly 1-dimensional plots.

Editing operations are possible for plots both in graphical and in text format. In the former case, the cursor will be used to set, for instance, the new value for a data point. In the latter case, the User will type the new value in a small dialog shell.

- **Add Point** : a new point will be added between the plot point closest to the cursor and the next one. The y value will be linearly interpolated from the previous and the next point.

- **Add Change** : as before, but the User will have now control over the y value of the new point (either using the cursor, in graphical mode, or inputting the value, in text mode).
- **Change Point** : the User will be able to modify the y value of the selected data point.
- **Delete Point** : the selected point will be deleted.
- **Bump Vector** : the y values of the point selected and of the next one will be incremented or decremented by a given value (controlled either by the cursor, or by text input).
- **Flat Vector** : the y values of the point selected and of the next one will be set to a given value (controlled either by the cursor, or by text input).
- **Zero Function** : the y values of all the data points of the plot (possibly excluding the first and the last ones) will be set to zero.
- **Flat Function** : the y values of all the data points of the plot (possibly excluding the first and the last ones) will be set to a given value.
- **Raise In Limits** : lets the User select two points in the plot (defining a range). then the User sets a new y value for the second point selected. To all the other points in the range the same modification will be applied.
- **Delete Points** : will delete all the points in the range specified by the User.
- **Align Points** : lets the User select two points in the plot (defining a range). Then all the y values of the points included in the range will be modified to lie on a straight line between the two limit points (linear interpolation).
- **Extend Change** : after changing the y value of one point, the User can extend the change to a range of points, by selecting a second point. The change applied to the first point will be propagated to all the points of the range in a linearly decreasing way (so that the y value of the second point will not be modified). The same operation can be carried on again and again, by selecting another “second point” (the first point being still the one used in the previous operation).

- The **LOAD/SAVE** Pulldown Menu

  The previous versions of the Xdataviewer needed to be started in association with an already configured data area (for the standalone Xdataviewer, a MOPS in an Unix Shared Memory Segment), and to use that data area during the all Xdataviewer life. Version 6.4 provides more flexibility; it is possible to start a Xdataviewer without any data, and it is possible to change data area on line. It is also possible to load a data area from a file, where it was previously saved. More over, it is possible to switch back and forth between a MOPS-like data area, and a file data area. The options contained in this menu give access to these operations.

  - **From Mops** : use as data area the last Mops (or internal buffer, in case of Embedded Dataviewer) accessed by the Xdataviewer (if any).
  - **From File** : use as data area the last file accessed by the Xdataviewer (if any).
  - **Load Mops** : loads a data area from a MOPS. A File Selection widget will help in the selection of the file identifying the MOPS.
  - **Load File** : loads a data area from a file (containing the copy of a MOPS in binary format). A File Selection widget will help in the selection of the file.
  - **Load Sequence** : reads a Sequence file (containing a list of names of files loadable by the Xdataviewer) , load the first file of the Sequence, and goes into the special Sequence mode. The previous four option names will be modifies into **NEXT** (to load the next file of the Sequence), **PREVIOUS** (to reload the previous file), **SHOW** (to display the list of the files contained in the Sequence, and to let the User select one), and **QUIT SEQUENCE** (to go back to the normal mode).
  - **Save To File** : enable the User to save into a file the complete contents of the current data area space used by the Xdataviewer. The file can later be reloaded by the Xdataviewer. A flag specifying if the file can be later modified by the Xdataviewer can also be

set. The selection of the file name is helped by a dialog shell. The User can also write a short remark, which will go in a file with the same name as the one used to save the data, plus a "*.remark*" extension. Data area already contained in a file cannot be saved again.

- **Source ?** : displays a temporary label, showing which data area is being currently used. The label is removed by entering and exiting it with the cursor.

- The **HELP** Pulldown Menu

  Using the options contained in this menu, it is possible to get on-line help both on the Xdataviewer itself, and on the meaning of the data displayed (this should be contained in a file provided by the Application, as explained more in detail in chapter "The Help Facility for the Xdataviewer"). By selecting an option, the corresponding information will be displayed in a scrolled text window contained in a temporary shell. This can be removed by entering and exiting it with the cursor.

- The **SELECT** LIST OF APPLICATION DEPENDENT OPTIONS

  When the User clicks on this button, a popup shell will show a scrolled list containing the names of all the Application dependent options (specified by *dv_set_selection_menu*) . By clicking on one of these names, the option will be chosen, a signal will be sent to the Application to notify it about the selection, and a message (set by dv_set_selection_prompt) will be displayed in the User Info Message Area, to explain the User what to do next. The popup shell will also disappear automatically when entered and then exited by the mouse cursor.

THE THIRD LINE OF THE INTERFACE

This line contains several Option Menus (PLOT, GRID, ZEROLINE, OP MODE, ZOOM ACTION, ZOOM TYPE). The entries contained in the corresponding popup menus can be used to modify the look of the displayed data. Only the latest Option Menu selected is active at any time, and can be recognized by its colour red.

- The **PLOT** Option Menu

  Using the options in this menu it is possible to switch between graphical and text data representation. in graphical mode, all the plots in a graph are always drawn. In text mode, the User can select if listing only one plot, or all the plots in the graph.

  - **Plot** : after selecting this option, click on one or more graphs. The data there contained will be redrawn in graphical format.

  - **List Plot** : after selecting this option, move the cursor on a graph, and position it close to a point of the plot to be listed (the name of the plot will be displayed on the cursor line). Click now, and the selected plot will be listed in text format on the graph window.

  - **List Graph** : after selecting this option, click on one or more graphs. The plots attached to the graphs will be listed in text format.

- The **GRID** Option Menu

  The options in this menu control the appearance of the graph borders. After selection of one of the options, click on one or more graphs to activate the specified setting.

  - **Grid OFF** : No grid will be drawn. The left and the top borders will display the x,y limits of the graph window.

  - **Grid ON** : Tics will be drawn around the graph. Together with some of them, also the x or y coordinate will be displayed. The system adapts by itself to the data range and to the number of pixels available for the graph, so that tics will always correspond to meaningful values.

  - **Grid FULL** : A subset of the above tics will be drawn, together with horizontal and vertical lines crossing the graph.

- The **ZEROLINE** Option Menu

  The options in this menu control the drawing of x and y axis on the graph. After selection of one of the options, click on one or more graphs to activate the specified setting.

  - **Zeroline OFF** : no axis will be drawn.

- **Zeroline ON** : the x axis will be drawn.
- **Zeroline XY** : both axis will be drawn.

- The **OP MODE** Option Menu

  Via the following options the User can specify if an operation performed on one graph should be also performed on all the other graphs, or not.

  - **OP ONE** : the operation (grid, zoom, cursor value display) will only be performed on the graph hosting the cursor.
  - **OP ALL** : the operation will be performed on all graphs at the same time.

- The **ZOOM ACTION** Option Menu

  This menu enables the User to perform zoom operations on the specified graph(s). The Xdataviewer system maintains a zoom history for each graph of the current view, so it is always possible to revert a series of zoom operations. The type of the zoom (*box, horizontal, vertica*l) is determined by the value of the next option menu (ZOOM TYPE), and will determine the kind of cursor (*cross, vertical line, horizontal line*).

  - **Zoom In** : lets the User select an area in a graph (by clicking on two x,y points to define the zoom region), and sets the *x,y visible window* of the graph to cover the selected area. If the zoom type is *box*, the two points will define a zoom rectangle. If the type is *horizontal* or *vertical*, the zoom will only affect the corresponding axis.
  - **Zoom Out** : it will double the range of the visible window around its centre. Both *x* and *y* ranges will be doubled if the zoom type is *box*, only *x* if type is *horizontal*, only *y* if type is *vertical*.
  - **Zoom Back Once** : reverts the last zooming operation performed on the graph.
  - **Zoom Back Orig** : goes back to the original *x,y* window displayed last time the view was drawn (controlled by *dv_grlimits*)
  - **Zoom Back Data** : goes back to a *x,y* window capable of accommodating all the data points of all the plots attached to the graph.

- The **ZOOM TYPE** Option Menu

  These options control the type of zoom to be used. The consequences have just been explained above.

  - **Box**
  - **Horizontal**
  - **Vertical**

5.3 The Cursor Line

As mentioned before, a line of information is displayed below each graph, and its contents is determined by the position of the cursor in the graph. If the horizontal size of the graph in pixels is large enough, the following information will be displayed :

- IN THE LEFT PART OF THE LINE :

  The **x value** of the data point closest to the cursor

  The **y value** of the data point closest to the cursor

  The **difference** between the y position of the cursor and the y value of the data point

  The **label** associated with the data point by *dv_pltxt* (if any)

- IN THE RIGHT PART OF THE LINE :

  The **x position** of the cursor

  The **y position** of the cursor

  The **name of the plot** to which the data point closest to the cursor belongs (useful if more than one plot are attached to the graph)

When the graph is represented in text format, the line will just contain the name of the selected plot.

6

The Embedded Dataviewer

A special version of the Xdataviewer running as a part of a User Application (instead than as a separate process) is now available. The integration of the Embedded Dataviewer within an Application Program is very straightforward if the XWindow Interface code generator "Xcreator" is used[5][6]. The integration is also possible without using Xcreator, but in this case one should follow very carefully the directions given in the section "EMBEDDED_DATAVIEWER without XCREATOR".

The main difference between the Embedded Dataviewer and the Stand Alone one is the way the communication between the Application Part and the Dataviewer is dealt with. In the Stand Alone case, the Application used signals to tell the Dataviewer that the data had changed, so that the Dataviewer could refresh its image. The signal was sent when the Application called the Dataviewer routine **dv_kick**(). In the Embedded case, the Application and the Dataviewer are parts of the same process, so that a normal call to a new Embedded Dataviewer Function (**dv__kick**) will play the same role played by the signal before. To distinguish between the two functions, the name of the Embedded Function contains two consecutive underscore characters.

In some cases, the Application also wanted to be informed when the User had selected some option or some data point on the Dataviewer. This was done by the Dataviewer signalling the Application using the signal SIGUSR1. In this case the Application had explicitly defined an interrupt handling routine to deal with this signal. This mechanism has been kept unmodified by the Embedded Dataviewer.

6.1 Callable functions specific to the Embedded Dataviewer

These routines and their arguments are described in details also in chapter 3, section "Embedded Dataviewer Specific Routines"

The user may pass arguments to the Dataviewer by using the following routines :

BEFORE THE DATAVIEWER CREATION : one of the following 4 routines to declare the data source

- **dv__sharename**(mopsname) to declare the name of the shared memory to be displayed by the Dataviewer
- **dv__pointer**(memorypointer) if the data are contained in the internal memory of the process
- **dv__filename**(filename) if the data are contained into a file (in MOPS format) (or on the Web)
- **dv__sequencename**(sequencename) if a sequence of files has to be used

One can also start the Embedded Dataviewer with no data : to do this use **dv_argument("empty","")**

The following routines can also be used, to set an initial view and graph.

- **dv__initialview**(viewname) to set a initial view to be displayed
- **dv__initialgraph**(graphname) to set an initial graph to be displayed

AFTER THE DATAVIEWER CREATION AND BEFORE THE APPLICATION MAIN LOOP

- **dv__argument**(argumentname,valuestring) (1 call per argument)
- **dv_before_main**() to get the Dataviewer ready

LATER

- **dv__kick**() to indicate the Dataviewer that the data to be displayed have changed
- **dv__kick_and_save**() similar to dv__kick(), but the contents of the view are saved back to the MOPS before the Dataviewer updates the display. Useful when the MOPS objects associated with a plot have to be changed synchronously by the User (editing) and by the Application. For example,

when the Application wants to change the MOPS colour object associated with a plot, to reflect the changes to the data object due to the editing operations.

AT ANY MOMENT AFTER THE DATAVIEWER CREATION

- **dv__gimme_window**() will return the XWindow Id of the Window containing the Dataviewer.

If the programmer wants TO CHANGE MOPS OR MEMORY POINTER without leaving the program, call again *dv__sharename* , *dv__pointer* , *dv__filename* or *dv__sequencename*, and then

- **dataviewer_initialize**()

Notice that the routines *dv__sharename, dv__initialview, dv__pointer, dv__argument* and *dv__kick* contain two consecutive underscore characters.

EXAMPLES are to be found in directories

/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/test_embed

6.2 How to clean the memory internally allocated by the Embedded Dataviewer

When the Internal Buffer is used (routine *dv__pointer*), the Application Programmer might want to have more than one buffer containing data, and might want to pass from one to the other buffer. In order to do things properly and not to leak memory, the following routine should be used when temporarily quitting an Internal buffer :

- **dv__clearchangeptr**()

When an Internal Buffer allocated via *malloc* has to be dropped definitively, the following routine shall be used :

- **dv_freeptr**(ptr) (where **ptr** is the pointer to the buffer)

6.3 A sequence of calls to create and initialize the Embedded Dataviewer

The following lines have been extracted from file *junky.c* in the directory /user/morpurgo/XSTUFF/DATAVIEWER_V6.4/test_embed . Xcreator Users may find more useful to look directly at the *junky.x* and *junky.user* files.

```
main(argc,argv)
.....
/* INFORMING THE DATAVIEWER ABOUT THE MOPS TO BE USED */
dv__sharename("@/user/morpurgo/DATAV/tests/kate");        /* or use dv__pointer() */
dv__initialview("view6");     /* if you want to set an initial view */
.....creation of the interface and in particular of the Widget
..... which should contain the Embedded Dataviewer
/* THE CREATION OF THE EMBEDDED DATAVIEWER  AS A CHILD OF A FRAME WIDGET (DATAVIEWER_FRAME)  : also next line is automatically generated by XCREATOR */
CreateSubtree_subtreexdataviewer("dataviewer_frame",dataviewer_frame,True);
/* REALIZATION OF THE ALL INTERFACE  (still inserted by XCREATOR)*/
XtRealizeWidget(junky);
/* THE FOLLOWING SECTION IS ONLY NEEDED IF YOU NEED TO BE INTERRUPTED BY THE DATAVIEWER PART (Menus EXTERNAL and SELECT) */
q = c_sdacc(@/user/morpurgo/DATAV/tests/kate,SD_WRITE);
dv_set_update_mode(DV_UPDATE_VIEW, q);
app_processid = getpid();
dv_set_app_processid(app_processid, q);
c_sdquit(q);
```

```
/* END OF THE SECTION */
/* PASSING SOME ARGUMENTS TO THE DATAVIEWER */
dv__argument("WHITE","");
dv__argument("UNIFORM","");
.........
dv_before_main();

/* ANOTHER LINE ONLY NEEDED WITH INTERRUPTS (YOU DECLARE YOUR INTERRUPT
HANDLER ROUTINE */
signal(SIGUSR1, increment_repeat_ec);
/*-----------------------------------------------------------------*/
/*              APPLICATION MAIN LOOP        (code generated by XCREATOR)  */
/*-----------------------------------------------------------------*/
for(;;) {
        XtMainLoop();
        if (user_signal == 0) break;
}
```

What an XCREATOR User has to insert in the .x file to generate such a code :

- declare a container widget (for instance a "frame") to contain the Dataviewer Interface

  ```
  WIDGET APPLICATION junky NULL
  WIDGET FORM     contains_all junky
  ...................
  * THE NEXT LINE DECLARES THE CONTAINER FOR THE DATAVIEWER
  WIDGET FRAME    dataviewer_frame contains_all
  ...................
  ```

- associate the Embedded Dataviewer creation routine (contained in library dv_embex.a) with the previously declared container. The next line will generate the CreateSubtree... call in the .c file.

  ```
  LINK_SUBTREE dataviewer_frame subtreexdataviewer
  ```

6.4 Embedded Dataviewer without Xcreator

This section is addressed to the Application Programmers who want to embed the Dataviewer in an XWindow Application whose code was NOT generated by XCREATOR. The operations shown in the previous section have to be executed in a given order, and problems may arise when a tool (like Face-Maker, or XDesigner) is used to generate the interface, because, depending on which part of the code the tool is generating, it might not be very simple to insert the right statements at the right places.

Let us assume the case in which the Embedded Dataviewer has to be positioned in a Frame Widget, which is contained inside a Form Widget. The Frame will be called *dataviewerFrame*, and the Form *Big-Form*. We assume that the Form has been created previously (possibly by Xdesigner, or FaceMaker), while we want to explicitly create the Frame .

- 1) The Part of the Interface which will contain the Embedded Dataviewer has to be created (and may be realized)
- 2) The Embedded Dataviewer, before its creation, has to know which mops or memory area has to use : this is done by calling one of *dv__sharename*() , *dv__pointer*(), *dv__filename*(), *dv__sequencename*() .
- 3) NOW WE CREATE the dataviewer frame : (the attachments are just an example)

  dataviewerFrame = XtVaCreateWidget("dataviewerFrame",xmFrameWidgetClass, **BigForm**, XmNtopAttachment,XmATTACH_FORM, XmNbottomAttachment, XmATTACH_FORM, XmNrightAttachment,XmATTACH_FORM, XmNleftAttachment,XmATTACH_FORM,NULL);

- 4) NOW WE CREATE the Embedded_Dataviewer
  CreateSubtree_subtreexdataviewer(“dataviewerFrame”,**dataviewerFrame**,True);
- 5) NOW WE MANAGE the Frame containing The Embedded Dataviewer
  XtManageChild(**dataviewerFrame**);
- 6) NOW WE INITIALIZE the Embedded Dataviewer.
  dv_before_main();
- 7) The Application reaches its Main Loop.

Points 2) to 6) can indeed be grouped inside a single subroutine, which could be invoked between the Realization of the Interface and the Application Main Loop

```
Create All Widgets(Toplevel and all its children)
XtRealizeWidget (Toplevel);
create_dataviewer(); /* routine grouping points 2) to 6) */
XtAppMainLoop (app_context);
```

Things may be slightly different, depending on the tool used to generate the interface.

6.5 Libraries to link

The following lines are taken from the makefile for the junky program

```
CFLAGS = -Aa -I /usr/include/X11R5 -I /usr/include/Motif1.2 -I/usr/opt/MOPS/include\
-D_CLASSIC_ANSI_TYPES -D_INCLUDE_HPUX_SOURCE -D_INCLUDE_POSIX_SOURCE \
-D_INCLUDE_XOPEN_SOURCE -D_INCLUDE_XOPEN_SOURCE_EXTENDED
LDFLAGS =   -L /usr/lib/X11R5 -L /usr/lib/Motif1.2 -L /usr/opt/MOPS/lib
junky : junky.o dv_embex.a
        fort77 -o junky junky.o dv_embex.a  ${LDFLAGS} -lshm\
        /user/morpurgo/XSTUFF/lib/Xcreator_lib.o\
        /user/morpurgo/XSTUFF/lib/Xcreator_enter_leave.o\
        -lXm -lXt -lX11 /lib/libm.a
# LINKING WITH THE fort77 OPTION  IS NEEDED TO FIND SOME OF THE LIBRARIES NEEDED BY
THE MOPS
junky.o : junky.c
#THE FOLLOWING IS FOR XCREATOR USERS
junky.c : junky.x junky.user
        Xcreator junky
```

For convenience, the two **.o** files from */user/morpurgo/XSTUFF/lib* have also been grouped in the library file **X_cr.a**, both in directory */user/morpurgo/XSTUFF/lib* and in directory */user/bim/XCREATOR/lib* .

7

Interactions between the Xdataviewer and the Application Program

There are several ways in which the Xdataviewer and an Application can interact.

- **No real interaction**.

  In the simplest case an Application can create a shared memory area, then the Xdataviewer can be run to display its contents. There is no real interaction here, apart from the sharing of the data.

- **Application -> Xdataviewer interaction.**

  The simplest interaction occurs when the Application , after creating or accessing the data area that the Xdataviewer has to display, periodically modifies it. This is typical of data acquisition programs, or of data analysis programs which can access different data sets. To make the Xdataviewer always displaying the most recent data, the Application will inform it that there is new data. This is done through a Unix signal when the Xdataviewer runs as a standalone process, and via a direct call to the refresh routine when the Dataviewer is embedded in the Application itself. This kind of interaction requires no special precaution by the Application Programmer, because the Xdataviewer is designed to deal with such a signal.

  This interaction is based on the use of routines like **dv_kick**, or **dv_kick_file** .

  An extension to this form of interaction occurs when the Application wants to force the Xdataviewer to replace the currently displayed data area with a new one. In this case, routine **dv_newdata_set** will be used (in combination with *dv_kick* or *dv_kick_file*). An example will be found in chapter 9.

- **Xdataviewer -> Application Interaction**

  A more complex case occurs when the Xdataviewer User wants to inform the Application that he has undertaken some action, relevant to the Application itself. For instance, the data displayed by the Xdataviewer could represent a LEP Orbit, and the User may want to disable some pickups, or to compute a correction for a portion of the Orbit. In this case, the Application must be informed about the intention of the User, and should also be able to know which data points have been selected by the User. It is clear that the Application must have been designed with this goal in mind, and that the options selectable by the User must have been defined by the Application (routine **dv_set_select_menu**), which will also contain the instructions to deal with these User requests. This kind of interaction has been again implemented using a Unix signal (this time from the Xdataviewer to the Application), and the Application will have a interrupt handling routine to deal with this signal. The Application will also be able to retrieve from the data area information explaining the reason for the signal (the option selected by the User) , and also the data point possibly selected (**dv_get_selection** , **dv_get_selection_cursor**).

  Notice that the Application has to call **dv_set_app_processid**, to write its process id in the data area, otherwise the Xdataviewer will not be able to signal it.

  Another case in which the Xdataviewer may want to signal the Application is when a plot has been edited by the User and rewritten into the data area, and the User wants to ask the Application to process the new data (e.g. the Function Editor). This is still implemented via a Unix signal, which can be triggered by using the second or the third option in the External Menu of the Xdataviewer interface (**write_data_send_signal** and **send_signal**).

- **Protection of the Application against unwanted interrupts**.

  Applications which do not want to be interrupted by the Xdataviewer do not need to declare an interrupt handling routine. They can either disable the Xdataviewer signalling capability (by not declaring any Application dependent option , and by disabling the signalling capabilities from the External Menu, via **dv_set_external_menu_string**), or, more simply, do not write their process id in the data area, so that the Xdataviewer cannot know which process it has to signal. The Application may also want to temporarily disable the Xdataviewer interrupting capability, by calling **dv_set_app_processid** with a 0 argument as process id, and resetting it to the right value when it can again accept interrupts.

8

Hints & Tricks (and frequently asked questions)

8.1 The typical structure of a Xdataviewer XWindow Application

The Application we want to describe is a process which has to set up a MOPS containing data to be displayed by the Xdataviewer and to be updated by the process itself as a consequence of User interaction (selection of new data, performing operations on the data, etc.)
We can identify the following sections in the source code :

- **MAIN BODY OF THE APPLICATION**
  - - Declaration of variables (among which mops pointer and mops object pointer)
  - - Process initialization (reading of arguments....)
  - - Creation of the Application User Interface
  - - Initialization of MOPS (creation, creation of objects) and of Xdataviewer layout (creation and tailoring of views, graphs, plots) . All of this will be contained in a single routine called, for instance, **mops_and_dv_initialize()** .
  - - Getting pointers to all the MOPS objects. I use **c_sdptr** to get pointer which I then treat as normal array pointers. All contained in a single routine (ex. **access_objects()**).
  - - Application Main Loop : here the process sits waiting for User interaction.

- **Routines to be called in response to actions of the User on the Application Interface**
  - - In particular, there will be routines which produce new data to be displayed by the Xdataviewer. These routines will invoke a routine called , for example , **kick_dataviewer()** . Inside this routine the Xdataviewer will be signalled, (**dv_kick**) , the MOPS reattached (**c_sdacc**) and the objects reaccessed (**access_objects**). The MOPS has to be reattached, and the objects reaccessed, because after *dv_kick* the MOPS pointer is no longer valid.

8.2 Hiding views from the View Selection list.

Many applications perform operations of different types, and many Views can be made available to the User to look at the different results. It might happen that, at some stage, some of these views contain inconsistent, or old, or misleading information. In this case the User should not have the possibility of accessing them. Previously the only solution was to delete these views using the *dv_delentry* routine, but this approach is clumsy, because it obliges the application to constantly delete and recreate views, and reattach graphs to them. A better approach is to use routines **dv_vwhide** and **dv_vwshow** to respectively remove one or more views from the selection list, or reinsert them in the list. This option gives the programmer a very simple way of always presenting the User with relevant information.

8.3 How much size should be allocated by the MOPS ?

The size to be allocated by the MOPS depends on three factors :

- - The number of objects to be created in the MOPS (*c_sdini*)
- - The individual size of each application defined MOPS object
- - The number of views, graphs and plots to be defined for the Xdataviewer. (*dv_init*)

The number of objects to be created in the MOPS, passed as parameter to *c_sdini*, defines the size used by the directory part of the MOPS. The Programmer should remember to add 5 to the number of Application specific data objects, because of the 5 objects created by *dv_init*. Besides the number of objects specified in *c_sdini*, the MOPS library itself will create other two directory slots. Each slot occupies, in the current implementation on HP-UX, 164 bytes. So the space requirement for this part is :

MOPS_directory_size = 164 x (number of objects specified in c_sdini + 2)

The individual size of each application defined MOPS object depends on the type and number of elements of the object itself. For instance, a double precision object of 1000 elements will occupy 8000 bytes (on HP-UX). It is up to the programmer to find out how much space is needed for his objects. The Programmer should not forget to account also for padding bytes, required when objects elements have to be aligned on specific bytes boundaries (ex. a numeric value cannot start on a odd byte, so mixing character and numeric variables in a structure may force the compiler to automatically insert padding bytes). We call the total space requirement for the Application objects the **Application_Object_Size** .

The size needed by the 5 Xdataviewer objects (DV_$VIEW etc.) can be computed using the routine *dv_space*. By calling

dv_space(no_views, no_graphs, no_plots, &Xdataviewer_space_needed) ,

Xdataviewer_space_needed will contain the size of the memory space needed by the Xdataviewer objects.
The size to be used when allocating the MOPS will then be

MOPS_directory_size + Application_Object_Size + Xdataviewer_space_needed

The Programmer should anyway not forget to allocate more space if the sizes of the data object may change (due to editing operations, for example).

8.4 Determining the initial representation mode of views, graph, plots.

- Use **dv_set_view** (or dv_set_view_and_graph) to control the initial view (and graph) selection.
- Use **dv_grlist** to have a graph initially listed.
- Use **dv_grlistmerge** to be able to list a graph containing plots with different x coordinates.
- Use **dv_pllist** to have a plot initially listed.
- Use **dv_set_default** (and similar routines) to modify the default creation values for graphs and plots.
- Use **dv_grsetgrid** to associate a grid to a graph.
- Do not forget to use **dv_set_update_mode** with the appropriated argument value to determine the behaviour of the Xdataviewer when data have to be redrawn.

8.5 dv_ObjectResize : a practical way of dealing with mops objects of variable size.

Many applications produce results contained in arrays whose size depends on parameters which can be changed by the User. For instance, in the LEP Qmeter, the User can specify the resolution of the instrument, so that a tune spectrum can consist of 512 or 1024 (or 2048) points. Or else, the User can start an operation like a Tune History, which produce one data point at each given time interval, and ask immediately for the data (few points available), or wait until the data buffer is full (3000 data points in our case). The problem is that these data arrays are contained in MOPS objects, and the Dataviewer plots, referring to these MOPS objects, will by default display the full contents of these objects. A possible solution is to manipulate the length of these objects using MOPS routines (*c_sdcut*, *c_sdaugm*) explained in [3]. Another solution is to drop these objects and recreate them each time with the right length (*c_sddrop*, *c_sdbook*).
These solutions are, in some respect, complex and not effective. The solution we propose is simpler and easier to code :

- At the initialization, define every object with its maximum number of elements to be possibly used by the Application.
- Every time new data to be put in the object is available, use routine **dv_ObjectResize** to redefine the “visible” size of the Object (its memory allocation will be unmodified) . This routine in fact modifies the field “number of elements” in a MOPS object’s directory entry, without modifying the object itself or performing any reorganization of the MOPS memory.
- Do not forget to resize also the auxiliary objects (labels, errorbars, ...) to keep consistency.

WARNING : this solution might be unusable if the User has to edit the Dataviewer data. In this

case the MOPS system might need to expand objects to accommodate new data points added by the User, and it can be cheated by the false information contained in the “number of elements” field.

8.6 Controlling the initial selection for the printers.

- The original default values for the Colour Printer, BW Printer and Text Printer are, respectively, “*pcrtxp*”, *not defined* and *not defined*.
- These values are overridden if the environment variables DV_COLORPRINTER, DV_BWPRINTER and DV_TEXTPRINTER are set.
- Those new values can be overridden by using the options -colorprinter <printername>, -printer <printername> and -textprinter <printername>.
- Finally, if some of the values are still undefined or not accessible after all of these checks, Xdata-viewer uses, in priority order, the values of the environment variables LPDEST, PRINTER, or the first printer in the list returned by the command “*lpstat -a*”.

9

Examples

9.1 A minimal example program

The following program creates a MOPS (it also creates the dummy key file just in case), books and fill one object, initialize the Xdataviewer part of the MOPS, creates one view, one graph and one plot (associated with the MOPS object previously defined), and attaches them together.
The size of the MOPS (30000) is arbitrary, as well as the number of objects (18) . Out of these 18 slots, 5 will be occupied by the Xdataviewer structures.

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"

main()
{
float *data;
int i, dv_status;
char *q;

/*---------- Create a MOPS ------------------------------*/
system("touch /usr/tmp/example1");
q = c_sdalloc("@/usr/tmp/example1",30000,SD_SHMALLOC);
c_sdini("anyname",18L,q);

/*---------- Create an object and fill it----------------*/
c_sdbook(100L,4L,"data","float",q);
data = (float *)c_sdptr("data",q);
for(i=0;i<100;i++) data[i] = i;

/*---------- Initialize the Dataviewer ------------------*/
dv_status = dv_init(1,1,1,q); dv_error(dv_status);

/*---------- Create a View ------------------------------*/
dv_status = dv_vwcreate("a view","View title", "",1,1,q);
dv_error(dv_status);

/*---------- Create a Graph and attach it to the View ---*/
dv_status = dv_grcreate("a graph","title","x axis","y axis",q);
dv_error(dv_status);
dv_status = dv_grattach("a graph","a view",0,0,0,0,q);
dv_error(dv_status);

/*---------- Create a Plot and attach it to the Graph ---*/
dv_status = dv_plcreate1("a plot",DV_HISTO,"data",q);
dv_error(dv_status);
dv_status = dv_plattach("a plot","a graph",q);
dv_error(dv_status);
}
```

After running the program, the MOPS will be left in the computer memory. The Xdataviewer can be used to look at the defined plot. Start it by typing :

Xdataviewer @/usr/tmp/example1

9.2 Minimal example without shared memory

This example is a copy of the previous one, with the only difference that the MOPS will be built inside an internal buffer (*mybuffer*) instead then in a shared memory segment. To avoid losing the MOPS when the program ends, routine *dv_save_file* will be invoked.
Instead of using a static buffer, one could dynamically allocate the space needed using *malloc* .

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"

static char mybuffer[30000];

main()
{
float *data;
int i, dv_status;
char *q;

/*---------------- Creates a MOPS without shared memory --------*/
q = mybuffer;
c_sdini("anyname",18L,q);

/*---------------- Create an object and fill it----------------*/
c_sdbook(100L,4L,"data","float",q);
data = (float *)c_sdptr("data",q);
for(i=0;i<100;i++) data[i] = i;

/*---------------- Initialize the Dataviewer -------------------*/
dv_status = dv_init(1,1,1,q); dv_error(dv_status);

/*---------------- Create a View -------------------------------*/
dv_status = dv_vwcreate("a view","View title", "",1,1,q);
dv_error(dv_status);

/*---------------- Create a Graph and attach it to the View ---*/
dv_status = dv_grcreate("a graph","title","x axis","y axis",q);
dv_error(dv_status);
dv_status = dv_grattach("a graph","a view",0,0,0,0,q);
dv_error(dv_status);

/*---------------- Create a Plot and attach it to the Graph ---*/
dv_status = dv_plcreate1("a plot",DV_HISTO,"data",q);
dv_error(dv_status);
dv_status = dv_plattach("a plot","a graph",q);
dv_error(dv_status);

/*---------------- Save the MOPS into a file ------------*/
dv_save_file("/usr/tmp/example1file",q);
}
```

After running the program, file */usr/tmp/example1file* will contain a copy of the MOPS. The Xdataviewer can be used to look at the defined plot. Start it by typing :

```
Xdataviewer  */usr/tmp/example1file
```

9.3 Data objects individually stored in files

This example shows how to store a data object into a file, and how to refer to it in the Xdataviewer routines (e.g. *dv_plcreate1*). The object used in the plot of the previous example will be stored in file "*/usr/tmp/fdata*" instead than in the MOPS. Finally, we show how to query and retrieve data from the Xdataviewer object file.

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"

static float fdata[100];   /* local storage for the object data */
main()
{
float *data;
int i, dv_status;
char *q;
long ln,lsiz,lcod;
int nelem;

/*---------------- Create a MOPS ------------------------------*/
system("touch /usr/tmp/example1tris");
q = c_sdalloc("@/usr/tmp/example1tris",30000,SD_SHMALLOC);
c_sdini("anyname",18L,q);

/*---fill an array with the data to be displayed, and save it as a Xdataviewer object file-----*/
for(i=0;i<100;i++) fdata[i] = i;
dv_file_array((char *)fdata,100,DV_FLOAT,"/usr/tmp/fdata");

/*---------------- Initialize the Dataviewer ------------------*/
dv_status = dv_init(1,1,1,q); dv_error(dv_status);
/*---------------- Create a View ------------------------------*/
dv_status = dv_vwcreate("a view","View title", "",1,1,q);
dv_error(dv_status);
/*---------------- Create a Graph and attach it to the View ---*/
dv_status = dv_grcreate("a graph","title","x axis","y axis",q);
dv_error(dv_status);
dv_status = dv_grattach("a graph","a view",0,0,0,0,q);
dv_error(dv_status);

/*---------------- Create a Plot and attach it to the Graph ---*/
/* REFER TO THE FILE CONTAINING THE DATA */
dv_status = dv_plcreate1("a plot",DV_HISTO,"*/usr/tmp/fdata",q);
dv_error(dv_status);
dv_status = dv_plattach("a plot","a graph",q);
dv_error(dv_status);

/*-----example on how to query and retrieve data from a Xdataviewer object file-----*/
dv_seek_object("/usr/tmp/fdata",&ln,&lsiz,&lcod);
printf("dv_seek_object %d      n %d   siz %d   cod %d\n",i,ln,lsiz,lcod);
nelem = dv_read_object(ffdata,100,"/usr/tmp/fdata");
for(i=0;i<nelem;i++) printf("%d : %f\n",i,ffdata[i]);
}
```

The Xdataviewer can then be started by the command :

Xdataviewer @/usr/tmp/example1tris

9.4 Example program to initialise and update the Xdataviewer

The following program creates the same MOPS structure as the previous one. When the MOPS creation is finished, it starts the Xdataviewer as a background process (from a script), and it refreshes the data every time the User types something.

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"

main()
{
float *data;
float a;
char dummy[30];
int i, dv_status;
char *q;

/*---------- Create a MOPS ------------------------------*/
system("touch /usr/tmp/example1");
c_sdkill("@/usr/tmp/example1");
q = c_sdalloc("@/usr/tmp/example1",30000,SD_SHMALLOC);
c_sdini("anyname",18L,q);

/*---------- Create an object and fill it----------------*/
c_sdbook(100L,4L,"data","float",q);
data = (float *)c_sdptr("data",q);
for(i=0;i<100;i++) data[i] = i;

/*---------- Initialize the Dataviewer ------------------*/
dv_status = dv_init(1,1,1,q); dv_error(dv_status);

/*---------- Create a View ------------------------------*/
dv_status = dv_vwcreate("view1","View title", "",1,1,q);
dv_error(dv_status);

/*---------- Create a Graph and attach it to the View ---*/
dv_status = dv_grcreate("a graph","title","x axis","y axis",q);
dv_error(dv_status);
dv_status = dv_grattach("a graph","view1",0,0,0,0,q);
dv_error(dv_status);

/*---------- Create a Plot and attach it to the Graph ---*/
dv_status = dv_plcreate1("a plot",DV_HISTO,"data",q);
dv_error(dv_status);
dv_status = dv_plattach("a plot","a graph",q);
dv_error(dv_status);

/*---------- Set the Xdataviewer update mode ------------*/
dv_status = dv_set_update_mode(DV_UPDATE_DATA,q);
dv_error(dv_status);

/*----quit the MOPS, to grant access to the Xdataviewer ------*/
c_sdquit(q);

/*---- start the Xdataviewer from a script        ------------*/
system("/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/testdoc/mystart");
```

```
for(;;) {
    q = c_sdacc("@/usr/tmp/example1",SD_RTC);
    data = (float *)c_sdptr("data",q);
    printf("type something :"); scanf("%s",dummy);
    for(i=0;i<50;i++) {a=data[i]; data[i] = data[99-i]; data[99-i]=a;}
    dv_kick(q);
}
}
```

The contents of the script **mystart** is :

```
../Xdataviewer @/usr/tmp/example1 view1 &
```

9.5 Refresh the Xdataviewer display using a MOPS in a file

In this example, the same data updating from the previous example is implemented using a file (in MOPS format), instead than a real Unix Shared Memory MOPS. The file */usr/tmp/examplefile1* is originally created by copying the previous MOPS into it. This operation could have been done outside this example program. Once the file exists, we will only work with it. The script *mystartfile* will start the Xdataviewer using the file as data source. There is a practical problem to be solved here : in some way the Xdataviewer must communicate to the Application its process id, otherwise the Application will not be able to kick the Xdataviewer. This problem is solved in the following way : when the Xdataviewer is started with a file as data source, it will write its process id in the DV_$COMMUNICATE object which must be found inside the file, and it will save the file itself. This explains why we want to start the Xdataviewer before entering the refreshing loop. The *sleep* instruction makes sure that the Xdataviewer has the time to write its process id into the file before this one is accessed again by the process.

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"

main()
{
float *data;
float a;
char dummy[30];
int i, dv_status;
char *q;

/*--- Attach the MOPS of previous example and save it to a file---*/
q = c_sdacc("@/usr/tmp/example1",SD_RTC);
dv_save_file("/usr/tmp/example1file",q);
c_sdquit(q);

/*---- start the Xdataviewer from a script       ------------*/
/*---- the Xdataviewer will write its pid into the file -------*/

system("/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/testdoc/mystartfile");
sleep(5);

/*---- attach the file, update the data, signal the Xdataviewer---*/
for(;;) {
    q = dv_attach_file("/usr/tmp/example1file");
    data = (float *)c_sdptr("data",q);
    printf("type something :"); scanf("%s",dummy);
    for(i=0;i<50;i++) {a=data[i]; data[i] = data[99-i]; data[99-i]=a;}
    dv_save_file("/usr/tmp/example1file",q);     /* to save the modification to the data array */
    dv_kick_file(q);
}
}
```

and the script **mystartfile**

```
../Xdataviewer */usr/tmp/example1file view1 &
```

9.6 Getting information back from the Xdataviewer

The following example shows how an application can obtain information back from the Xdataviewer. We will use the same MOPS previously created, to which we will add some Application Dependent options (using *dv_set_selection_menu*) . The program will declare a interrupt handler (routine **increment_repeat_ec**) which will be activated every time the Xdataviewer User selects an application dependent option (from the Select List), or clicks on a graph after such a selection, or uses the second and third options from the External Menu. The program will print the reason for the interrupt, together with the location of the cursor on the graphical part.

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"
#include <signal.h>

int increment_repeat_ec()

{
      char    *q;
      int dv_status;
      int processid, col;
      char    viewname[16], graphname[16], plotname[16],
      xdataname[16], ydataname[16], textname[16];
      int     option, index;
      double  xvalue, yvalue, xcursor, ycursor;
      char    textstring[16];


      signal(SIGUSR1, SIG_IGN);
      q = c_sdacc("@/usr/tmp/example1",SD_WRITE);
      dv_get_selection(&option, viewname, graphname, plotname,
                xdataname, ydataname, textname, &index, &xvalue,
                &yvalue, textstring, q);
      dv_get_selection_cursor(&xcursor, &ycursor, q);
      printf("option = %d\n", option);
      if (option == -DV_SIG)
            printf("signal\n");
      else if (option == -DV_WRT_AND_SIG)
            printf("write and signal\n");
      else
      {
            printf("viewname = %s\n", viewname);
            printf("graphname = %s\n", graphname);
            printf("plotname = %s\n", plotname);
            printf("xdataname = %s\n", xdataname);
            printf("ydataname = %s\n", ydataname);
            printf("textname = %s\n", textname);
            printf("index = %d\n", index);
            printf("xvalue = %f\n", xvalue);
            printf("yvalue = %f\n", yvalue);
            printf("textstring = %s\n", textstring);
            printf("cursor = %f %f \n", xcursor, ycursor);
      }
      dv_kick(q);
      signal(SIGUSR1, increment_repeat_ec);
}

main()
```

```
{
      int app_processid;
      char    *q;
char     anntext[10][16];

      q = c_sdacc("@/usr/tmp/example1",SD_WRITE);

      /* We will add some Application dependent options */
      /* to the MOPS previously created                 */
        strcpy(anntext[0],"server0");
        strcpy(anntext[1],"server1");
        strcpy(anntext[2],"server2");
        strcpy(anntext[3],"server3");
        strcpy(anntext[4],"server4");
        strcpy(anntext[5],"server5");
        strcpy(anntext[6],"server6");
        strcpy(anntext[7],"server7");
        strcpy(anntext[8],"server8");
        dv_set_selection_menu(9,anntext, q);
        dv_set_selection_prompt(8,"something different", q);

      dv_set_update_mode(DV_UPDATE_VIEW, q);
      app_processid = getpid();
      dv_set_app_processid(app_processid, q);
      c_sdquit(q);
/*---- start the Xdataviewer from a script       ------------*/
        system("/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/testdoc/mystart");

/*--- declare the interrupt handler */
        signal(SIGUSR1, increment_repeat_ec);
/*--- sit in a loop, waiting for an interrupt*/
        for (;;)   pause();  /* wait to be woken up by the Xdataviewer */
 }
```

Notice how **getpid** and **dv_set_app_processid** are used to let the Xdataviewer know the process id of the application process (to be signalled).

9.7 Application - Xdataviewer interaction using files

This example is similar to the previous one, but here there is no shared memory involved ! The interaction between the Application and the Xdataviewer is based on the updating of a binary file (previously formatted as a MOPS). Notice how **dv_attach_file** replaces *c_sdacc*, and how **dv_quit_file** is used, to release the buffer used to load the contents of the file. The process initially modifies the file by writing its process id and saving it, and then the Xdataviewer will modify the file every time it has to signal something to the application.

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"
#include <signal.h>
int increment_repeat_ec()

{
      char    *q;
      int dv_status;
      int processid, col;
      char    viewname[16], graphname[16], plotname[16],
      xdataname[16], ydataname[16], textname[16];
      int     option, index;
      double  xvalue, yvalue, xcursor, ycursor;
      char    textstring[16];


      signal(SIGUSR1, SIG_IGN);
      q = dv_attach_file("/user/morpurgo/MOPSFILES/mopsfile");
      dv_get_selection(&option, viewname, graphname, plotname,
                xdataname, ydataname, textname, &index, &xvalue,
                &yvalue, textstring, q);
      dv_get_selection_cursor(&xcursor, &ycursor, q);
      printf("option = %d\n", option);
      if (option == -DV_SIG)
            printf("signal\n");
      else if (option == -DV_WRT_AND_SIG)
            printf("write and signal\n");
      else
      {
            printf("viewname = %s\n", viewname);
            printf("graphname = %s\n", graphname);
            printf("plotname = %s\n", plotname);
            printf("xdataname = %s\n", xdataname);
            printf("ydataname = %s\n", ydataname);
            printf("textname = %s\n", textname);
            printf("index = %d\n", index);
            printf("xvalue = %f\n", xvalue);
            printf("yvalue = %f\n", yvalue);
            printf("textstring = %s\n", textstring);
            printf("cursor = %f %f \n", xcursor, ycursor);
      }
      dv_quit_file(q);
      signal(SIGUSR1, increment_repeat_ec);
      return(0);

}

main()
```

```
{
        int app_processid;
        char    *q;


        q = dv_attach_file("/user/morpurgo/MOPSFILES/mopsfile");
        dv_set_update_mode(DV_UPDATE_VIEW, q);
        app_processid = getpid();
        dv_set_app_processid(app_processid, q);
        dv_save_file("/user/morpurgo/MOPSFILES/mopsfile",q);
        dv_quit_file(q);
/*---- start the Xdataviewer from a script        ------------*/
system("/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/testdoc/mystartfile");
        signal(SIGUSR1, increment_repeat_ec);
        for (;;) pause();   /* wait to be woken up by the Xdataviewer */
}
```

The script *mystartfile* contains the line
../Xdataviewer */user/morpurgo/MOPSFILES/mopsfile &
to start the Xdataviewer using a file.

9.8 Changing the displayed data area space directly from the Application

In this example we show how the Application Program can force the Xdataviewer to replace the data area space to be displayed. The important routine in this example is **dv_newdata_set** . The new data area space, as well as the old one, can be contained either in a shared memory MOPS or in a MOPS formatted file. If the old data was contained in a file, after *dv_newdata_set* is called, and before the Xdataviewer is kicked, the old file has to be saved with *dv_save_file*, because the Xdataviewer will read from the file itself the name of the new data area space to be loaded. If the old data area space was a shared memory MOPS, this name is directly written by *dv_newdata_set* in a dedicated field of the MOPS.

```
#include <sys/types.h>
#include <sps.shm.h>
#include "../dataviewer.h"

main()
{
float *data;
float a;
char dummy[30];
int i, dv_status;
char *q;

char sharename[80];
char new_sharename[80];
int status, pid=0;

/*---- start the Xdataviewer from a script        ------------*/
system("/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/testdoc/mystartfile &");
sleep(10);
strcpy(sharename,"*/user/morpurgo/MOPSFILES/junk1");
/*---get Dataviewer pid----------*/
q = dv_attach_file("/user/morpurgo/MOPSFILES/junk1");
status = dv_get_dv_processid(&pid,q);
printf("status %d, dvpid %d\n",status,pid);
dv_quit_file(q);

/*---- loop : read the new data source name and transmit it to the Xdataviewer ---*/
for(;;) {
    printf("NEW DATA SOURCE ? ");scanf("%s",new_sharename);
    printf("new source : %s\n",new_sharename);

    if (sharename[0] == '*') { /* current data area space is a file */
          q = dv_attach_file((char *)&sharename[1]);
          dv_newdata_set(new_sharename,q);
           /* save the old file, from where Xdataviewer will read the new data source name */
          dv_save_file((char *)&sharename[1],q);
          dv_kick_file(q);
    } else {  /* current data area space is a MOPS */
          q = c_sdacc(sharename,SD_RTC);
          dv_newdata_set(new_sharename,q);
          dv_kick(q);
    }
    strcpy(sharename,new_sharename);
}
}
```

10

The picture drawing facility for the Xdataviewer

Until now, the Xdataviewer was only able to display data arrays, in graphical or text format. Sometimes adding other kind of graphical information to a picture may facilitate its comprehension. To this purpose a new facility, completely integrated in the current framework, has been developed. This note describes in detail this new facility.

10.1 General principles

- We define as a "**Picture**" a collection of graphical objects (points, lines, rectangles, circles, text strings etc.), together with colour and style definitions and any other instruction needed to produce the desired result on the screen.
- The data defining the Picture will be contained into a mops object (ex. "picture1"), as a sequence of codes and parameters. These codes and parameters will be used by the Xdataviewer program when the Picture has to be drawn.
- A new type of Xdataviewer plot , "DV_PICTURE", has been defined. The programmer will use this type to link the mops object containing the picture with a Xdataviewer plot . For example :

  dv_plcreate1(plotname,DV_PICTURE,"picture1",q)
- The plot will be attached to one or more graphs, in the standard way.
- Whenever a graph to which the plot is attached is displayed, the Xdataviewer will execute all the intructions contained in the "picture1" object to draw the corresponding picture.
- It will be possible to build a picture element by element, inserting in the Application calls to the different *dv_p_xxx* routines described later (10.5).
- It is also possible to specify the contents of a picture in a text file, using the language defined in (10.5.6), and to build the picture passing the file name as an argument to routine *dv_p_readfromfile*.

Note that

1. the coordinates of the picture elements will not be used for the automatic rescaling of the graph limits.
2. The Xdataviewer will draw the plots of type DV_PICTURE before drawing the other plots, not to hide the real data.
3. Plots of type DV_PICTURE will not be detected by the algorithm to find the data point closer to the cursor.

10.2 A short example

As a starting point, let us examine a simple example program (example1.c, in /user/morpurgo/XSTUFF/DATAVIEWER6.4/test_picture)

```
#include    <sys/types.h>
#include    <stdio.h>
#include    <sps.shm.h>
#include    "../dv_picture.h"
#include    "../dataviewer.h"
#include    <math.h>

#define PICTUREMOPS "@/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/test_picture/mops"
static char *q;

main(argc,argv)
int argc;
char **argv;
{
int i;
int status;
```

```
int dv_status;
int *dataptr;

/*========MOPS CREATION AND INITIALIZATION======*/
c_sdkill(PICTUREMOPS);
q = c_sdalloc(PICTUREMOPS,100000L,SD_SHMALLOC);
c_sdini("PICTURE TEST",30L,q);
c_sdzero(1000,1,"picture","char",q);
c_sdzero(20,4,"data","int",q);
dataptr = (int *)c_sdptr("data",q);
for(i=0;i<10;i++) dataptr[i] = i;
for(i=10;i<20;i++) dataptr[i] = 20-i;
c_sdquit(q);

q=c_sdacc(PICTUREMOPS,SD_RTC);

/*========   PICTURE CREATION=======*/
status = dv_p_cleancnt("picture",q);
dv_p_debug("picture",1,q);  /* OPTIONAL : ENABLES PRINTOUT */
                        /* OF DEBUG STATEMENTS         */
dv_p_linewidth("picture",4,q);
dv_p_color("picture",DV_GREEN,q);
/* NEXT LINE SPECIFIES ONE OF THE POSSIBLE STRATEGIES TO   */
/* ESTABLISH THE PIXEL COORDINATES OF THE PICTURE ELEMENTS */
dv_p_sizetype("picture",DV_P_PERCENTAGE,q);
dv_p_rectangle("picture",0.1,0.1,.8,.8,q);
dv_p_circle("picture",0.5,0.5,.3,q);
dv_p_point("picture",0.5,0.5,q);
dv_p_line("picture",0.0,0.0,1.0,1.0,q);
dv_p_text(“picture”,0.1,0,9,”Nice picture , isn’t it ?”,q);
dv_p_sizetype("picture",DV_P_PERCEPIXEL,q);  /* A DIFFERENT STRATEGY */
dv_p_color("picture",DV_VISIBLE,q);
dv_p_circle("picture",0.5,0.5,40.0,q);

dv_p_analyse("picture",q);  /* OPTIONAL : WILL DESCRIBE */
/* ON THE TERMINAL THE CONTENTS OF THE PICTURE          */

/*========SETTING UP DATAVIEWER VIEWS, GRAPHS, PLOTS ======*/
dv_init(5, 5, 5, q);
dv_status = dv_vwcreate("view1","junky", "", 1,1,q);
dv_status = dv_grcreate("graph1", "graph1","index","y",q);
dv_grattach("graph1", "view1",0,0,0,0,q);
dv_status = dv_plcreate1("plot1", DV_HISTO, "data", q);
dv_status = dv_plattach("plot1","graph1",q);
dv_status = dv_plcreate1("plot2", DV_PICTURE, "picture", q);
dv_status = dv_plattach("plot2","graph1",q);

dv_status = dv_vwcreate("view2","junky", "", 1,1,q);
dv_status = dv_grcreate("graph2", "graph2","index","y",q);
/* NEXT LINE IS NEEDED IF A GRAPH HAS ONLY PICTURE PLOTS */
/* ATTACHED TO IT */
dv_grlimits("graph2", 0.0, 30.0, 0.0, 30.0, q);
dv_grattach("graph2", "view2",0,0,0,0,q);
dv_status = dv_plattach("plot2","graph2",q);
c_sdquit(q);
}
```

This program creates two views (“view1” and “view2”). view1 contains a graph (“graph1”), to which two plots (“plot1” and “plot2”) are attached. plot1 is a normal histogram data plot, while plot2 contains a picture. Figure 10.1 shows what is displayed on the screen when view1 is selected from the Xdataviewer.

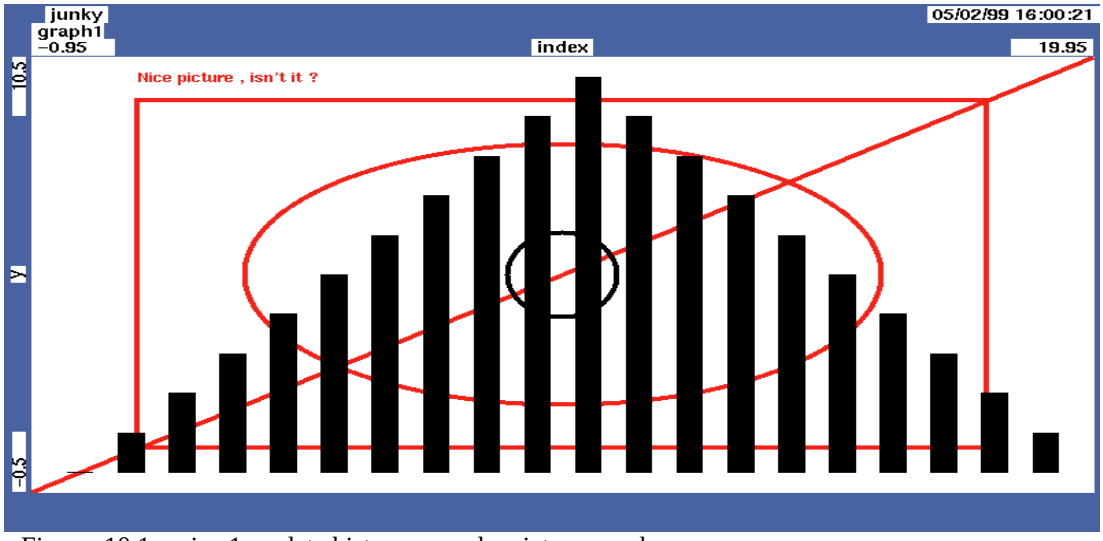

Figure 10.1 . view1 : a data histogram and a picture are shown.

The histogram and the picture are shown. Figure 10.2 shows view2, which only contains the picture.

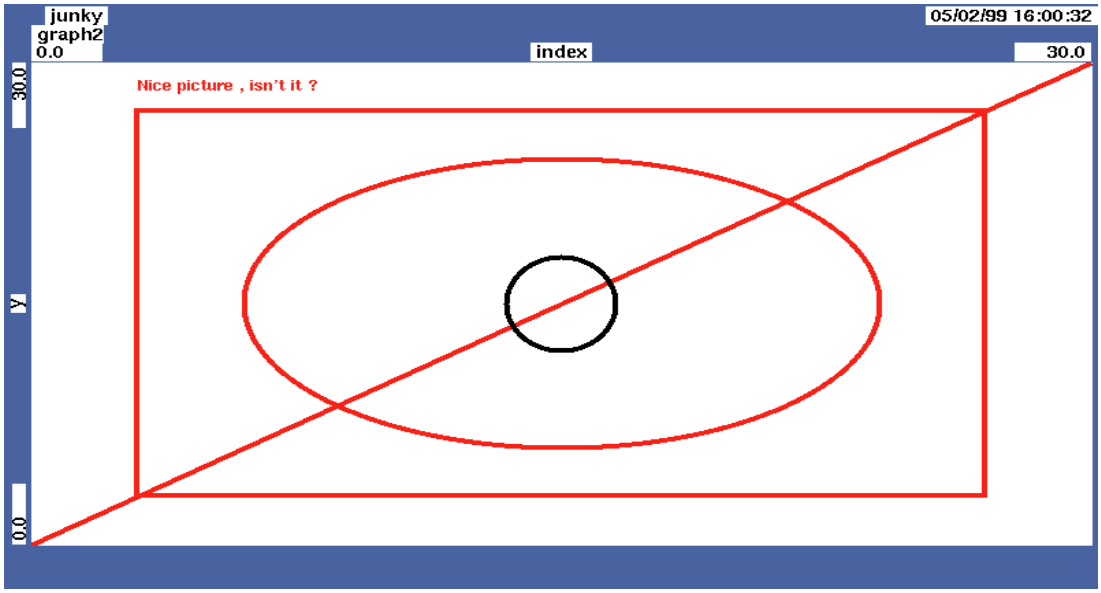

Figure 10.2 . view2 : only the picture is shown.

One thing is worth noticing : the central small circle maintains its proportions much better than the bigger one (clearly transformed in an ellipse). This depends on the choice of the strategy used for establishing the pixel coordinates of the picture elements. We will examine this problem in some detail in the next section.

By looking at the source code of the example program we also notice that we build our picture as a sequence of graphical elements (points, lines, rectangles, circles, etc.) interleaved with specifications of the way in which these elements have to be displayed (colours, linewidth, coordinate strategies, etc.).

The allocation of a mops object to contain the picture is convenient, because then the Xdataviewer can recognize the picture as any other plot, so that no major modification of the software structure is required.

10.3 Coordinate strategy

When a picture to be displayed by the Xdataviewer is conceived, one should be aware of the following things :

- The size of the graphical area where the picture is displayed may change (if the User rescales the Xdataviewer Window, or as a result of a Subview selection).
- The visible “data window” over the x,y plane can also change (as a result of data-driven graph auto-rescaling, or zooming, or shifting around with the arrow keys)

Many different application may impose different requirements to where and how a picture has to be drawn. For example, a picture might need to be attached at a specific point of the x,y plane, so that it has to be repositioned in case of zoom or change of the visible data window. It might require a fixed size (so that, for instance, a circle will remain a circle, or it might have to be resized according to the variable limits of the visible window.

It turns out that there is a natural independency between the way the position of an object (ex. the centre of a circle, the bottom left corner of a rectangle) and the way the object dimensions (ex. the ray of the circle, the sides of a rectangle) depend on the variability of the graphical area size and of the data window. Therefore we will provide an independent way of specifying this dependence.

To make it short, we foresee the following “strategies” :

- DV_P_DATASIZE : both position and size of an object are specified in x,y coordinates, and their translation in pixels will depend on the x,y limits of the visible window and on the size of the graph in pixels (a circle may become an ellipse). Of course, objects will not be displayed if their position lies out of the visible window.
- DV_P_PIXELSIZE : the position of an object is still specified in x,y coordinates, but the size is specified in pixels. Therefore the object will have fixed size, independently on the limits of the visible window and on the size of the graph (a circle will remain a circle).
- DV_P_PERCENTAGE : both position and size of an object are expressed as fractions of the total size of the graph. A position of 0.0,0.0 will specify the bottom left corner of the graph, a position of 1.0,1.0 the top right corner. This strategy is convenient if we want to display an object unregardingly from the x,y limits of the visible window.
- DV_P_PERCEPIXEL : in this case, the position of an object is expressed as a fraction of the total size of the graph (as for DV_P_PERCENTAGE), but the size if specified in pixels (as for DV_PIXELSIZE).

Our facility provides primitives for drawing different kind of objects, some of which only have positions (points, lines, arrows, segments, polygons), while others have positions and sizes (diagonals, rectangles, circles, arcs). The representation of these objects will depend on the strategy selected. When the dataviewer has to draw a picture, it sets the initial default strategy to DV_P_DATASIZE. The programmer can change the strategy at any time inside a picture by using the routine *dv_p_sizetype()* .

10.4 The internal structure of a picture

As we have seen in the example, a picture is contained into a Mops object, and it is built by a sequence of function calls. Each of these calls add one element or one instruction to the picture, filling some space inside the memory belonging to the Mops object. The first two bytes of the Mops object contain a counter, which is used at build time to know where to add the information for the next graphic element, and at display time to know which part of the Mops object effectively contains information to be displayed. Therefore, if we start counting from zero, the first graphic element or instruction will start at byte 2 of the Mops object. Each element occupies a specific number of bytes, described in the next section.

10.5 The Callable Interface

Let us examine in detail the functions available to the programmer to build a picture. These functions can be found in the *dv_picture.o* file. In all the functions, **objectname** contains the name of the Mops object, and **q** is the usual pointer to the Mops.

We can distinguish between different classes of functions.

10.5.1 Functions acting on the “picture counter”.

These functions modify or access the counter contained in the first two bytes of a picture.

int dv_p_cleancnt(char *objectname, char *q)

Resets the picture counter to zero : should be used when one wants to build a new picture reusing the same Mops object.

int dv_p_setcnt(char *objectname, short pos, char *q)

To be used with caution. Sets the picture counter to the value specified by **pos** . Useful when the programmer wants to modify the parameters of a picture element. For example, if byte 72 of the object is the starting point for a COLOR definition, and this colour has to be changed, the programmer will call

dv_setcnt(objectname, 72,q)
dv_p_color(objectname,newcolor,q)

and the previous colour definition will be replaced by this latter one.

int dv_p_querycnt(char *objectname, char *q)

Returns the current value of the picture counter.

int dv_p_queryptr(char *objectname, char *q)

Returns the current value of the picture counter, also setting it to 2 if it was zero. Mainly for internal usage.

10.5.2 Functions specifying instructions on how to draw the picture.

Each of these functions add an instruction to the picture object. This will influence the way all the picture elements defined after the instruction are drawn. For example, the call

dv_p_color(objectname,DV_GREEN,q)

will cause all the picture elements defined after it (and before the next COLOR instruction) to be drawn in green colour.

These functions return the value of the picture counter where their code starts, or -1 in case of error. The returned value can be used later, if one needs to modify some element in the picture (see “how to modify a picture element” in the Appendix).

int dv_p_color(char *objectname, int color, char *q)

Sets the foreground colour for all the followings picture elements. **color** can be any of the colours defined in the dataviewer include file *dataviewer.h* (including DV_VISIBLE and DV_INVISIBLE). It occupies 4 bytes.

int dv_p_backcolor(char *objectname, int color, char *q)

Sets the background colour (used as text background by the *dv_p_textimage* routine). **color** can be any of the colours defined in the dataviewer include file *dataviewer.h* (including DV_VISIBLE and DV_INVISIBLE). It occupies 4 bytes.

int dv_p_linestyle(char *objectname, int style, char *q)

Sets the line style for all the following picture elements. **style** can be one of DV_P_LINESOLID (default), DV_P_LINEONOFF, DV_P_LINEDOUBLEDASH. It occupies 4 bytes.

int dv_p_linewidth(char *objectname, int width, char *q)

Sets the line width for all the following picture elements. By default, the line width is 0, which corresponds to a 1-pixel thick line. It occupies 4 bytes.

int dv_p_marker(char *objectname, int marker, char *q)

Sets the marker type for all the following picture elements of type point. **marker** can be any of the marker types defined in the dataviewer include file *dataviewer.h* . It occupies 4 bytes.

int dv_p_sizetype(char *objectname, int type, char *q)

Sets the strategy for computing the pixel coordinate for all the following picture elements (as described in detail in the previous section). **type** can be one of DV_DATASIZE (default), DV_PIXELSIZE, DV_PERCENTAGE, DV_PERCEPIXEL. It occupies 4 bytes.

int dv_p_fontsize(char *objectname, int size, char *q)

Sets the font size for the following text elements contained in the picture. **size** can be one of DV_P_SMALLFONT, DV_P_LARGEFONT. It occupies 4 bytes.

10.5.3 Functions to add graphical elements to the picture.

Each function will return the value of the picture counter where its code starts, or -1 in case of error.

int dv_p_point(char *objectname, double x, double y, char *q)

Adds a picture element of type *point*. Its positions will be **x** and **y**, and will be translated into pixel position accordingly to the selected strategy (see previous section). At run time, it will be drawn as a marker of the type specified in the latest *dv_p_marker* call executed. It occupies 18 bytes.

int dv_p_points(char *objectname, short npoints, double *xarray, double *yarray, char *q)

Adds **npoints** picture elements of type *point*. **xarray** and **yarray** contain the positions of each point, to be translated into pixel position accordingly to the selected strategy (see previous section). At run time, each point will be drawn as a marker of the type specified in the latest *dv_p_marker* call executed. It occupies 4+npoints*16 bytes.

int dv_p_pointscol(char *objectname, short npoints, double *xarray, double *yarray, short *colarray, char *q)

Adds **npoints** picture elements of type *point*. **xarray** and **yarray** contain the positions of each point, to be translated into pixel position accordingly to the selected strategy (see previous section). **colarray** contains the colour code for each point. At run time, each point will be drawn as a marker of the type specified in the latest *dv_p_marker* call executed. It occupies 4+npoints*18 bytes.

int dv_p_line(char *objectname, double x1, double y1, double x2, double y2, char *q)

Adds a picture element of type *line*. **x1**,**y1**, **x2**,**y2** define the endpoints of the line. It occupies 34 bytes.

int dv_p_arrow(char *objectname, double x1, double y1, double x2, double y2, char *q)

Adds a picture element of type *arrow*. **x1**,**y1**, **x2**,**y2** define the endpoints of the line. It occupies 34 bytes.

An arrow is often useful to focus the reader attention on a particular point of the graph.

int dv_p_lines(char *objectname, short npoints, double *xarray, double *yarray, char *q)

Adds **npoints-1** *connected lines* (the end point of one line will be used as start point for the next one). **xarray** and **yarray** contain the coordinates of the endpoints. Elements 0 and 1 of these array will be used as endpoints for line 1, elements 1 and 2 for line 2 etc. It occupies 4+16*npoints bytes.

int dv_p_linescol(char *objectname, short npoints, double *xarray, double *yarray, short *colarray, char *q)

Same as *dv_p_lines*, but each line may have a different colour, specified by **colarray**. It occupies 4+18*npoints bytes.

int dv_p_segments(char *objectname, short npoints, double *x1array, double *y1array, double *x2array, double *y2array, char *q)

Adds **npoints** *non-connected lines.* **x1array**, **y1array**, **x2array**, **y2array** contain the endpoints for the lines. It occupies 4+32*npoints bytes.

int dv_p_segmentscol(char *objectname, short npoints, double *x1array, double *y1array, double *x2array, double *y2array, short *colarray, char *q)

Same as *dv_p_segments*, but each line may have a different colour, specified by **colarray**. It occupies 4+34*npoints bytes.

int dv_p_diagonal(char *objectname, double x, double y, double width, double height, char *q)

Adds a picture element of type *diagonal*. A diagonal is a line whose coordinates are specified by a starting point (**x**, **y**) and by a **width** and a **height**. In this way the translation into pixel cooordinates will be consistent with the translation of the coordinates of a rectangle, so that this type can be used to represent its diagonal. It occupies 34 bytes.

int dv_p_diagonals(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, char *q)

Adds **npoints** *diagonals*. **xarray**, **yarray**, **widtharray**, **heightarray** specify the starting point coordinates and the extension of each diagonal. It occupies 4+32*npoints bytes.

int dv_p_diagonalscol(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, short *colarray, char *q)

Same as *dv_p_diagonals*, but each diagonal may have a different colour, specified by **colarray**. It occupies 4+34*npoints bytes.

int dv_p_rectangle(char *objectname, double x, double y, double width, double height, char *q)

Adds a picture element of type *rectangle*. **x** and **y** represent the coordinates of the bottom left corner of the rectangle, and **width** and **height** its extension. It occupies 34 bytes. Only the contour of the rectangle will be drawn.

int dv_p_rectangles(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, char *q)

Adds **npoints** *rectangles*. **xarray**, **yarray**, **widtharray** and **heightarray** respectively contain the bottom left corner coordinates and the extensions of the npoints rectangles. It occupies 4+32*npoints bytes.

int dv_p_rectanglescol(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, short *colarray, char *q)

Same as *dv_p_rectangles*, but each rectangle may have a different colour, specified by **colarray**. It occupies 4+34*npoints bytes.

int dv_p_rectanglefill(char *objectname, double x, double y, double width, double height, char *q)

Same as *dv_p_rectangle*, but here the rectangle is filled. It occupies 34 bytes.

int dv_p_rectanglesfill(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, char *q)

Same as *dv_p_rectangles*, but here the rectangles are filled. It occupies 4+32*npoints bytes.

int dv_p_rectanglescolfill(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, short *colarray, char *q)

Same as *dv_p_rectanglescol*, but here the rectangles are filled. It occupies 4+34*npoints bytes.

int dv_p_polygonfill(char *objectname, short npoints, double *xarray, double *yarray, char *q)

Adds a picture element of type *filled polygon*. A polygon will be the inside of a set of connected lines, whose endpoints are specified by **xarray** and **yarray** (like in *dv_p_lines*). The polygon will be automatically closed , even if the first and the last point do not coincide. It occupies 4+16*npoints bytes.

int dv_p_circle(char *objectname, double x, double y, double ray, char *q)

Adds a picture element of type *circle*. **x** and **y** are the coordinates of the centre, and **ray** is the ray of the circle. It occupies 26 bytes. Only the circle contour will be drawn.

int dv_p_circles(char *objectname, short npoints, double *xarray, double *yarray, double *rayarray, char *q)

Adds **npoints** *circles*. **xarray** and **yarray** contain the coordinates of the centres, and **rayarray** the rays of the circles. It occupies 4+24*npoints bytes.

int dv_p_circlescol(char *objectname, short npoints, double *xarray, double *yarray, double *rayarray, short *colarray, char *q)

Same as *dv_p_circles*, but each circle may have a different colour, specified by **colarray**. It occupies 4+26*npoints bytes.

int dv_p_circlefill(char *objectname, double x, double y, double ray, char *q)

Same as *dv_p_circle*, but here the circle is filled. It occupies 26 bytes.

int dv_p_circlesfill(char *objectname, short npoints, double *xarray, double *yarray, double *rayarray, char *q)

Same as *dv_p_circles*, but here the circles are filled. It occupies 4+24*npoints bytes.

int dv_p_circlescofill(char *objectname, short npoints, double *xarray, double *yarray, double *rayarray, short *colarray, char *q)

Same as *dv_p_circlescol*, but here the circles are filled. It occupies 4+26*npoints bytes.

int dv_p_arc(char *objectname, double x, double y, double width, double height, double angle1, double angle2, char *q)

Adds a picture element of type *arc*. An arc is defined by a center (**x**,**y**), the sizes of a rectangle which can inglobe it (**width**, **height**), a starting angle (**angle1**) and an angular width (**angle2**) expressed in degrees. The angles will be internally rounded to 1/64 of a degree, following the XWindow convention. *angle1* represents an angular offset from the three-o-clock position, and *angle2* the offset from *angle1*. Positive values indicate counterclockwise motion, negative values clockwise motion. It occupies 50 bytes.

int dv_p_arcs(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, double *angle1array, double *angle2array, char *q)

Adds **npoints** *arcs*. **xarray** and **yarray** contain the coordinates of the centres, **widtharray** and **heightarray** the sizes of the inglobing rectangles, **angle1array** and **angle2array** the starting angles and the angular widths of each arc. It occupies 4+48*npoints bytes.

int dv_p_arcscol(char *objectname, short npoints, double *xarray, double *yarray, double

***widtharray, double *heightarray, double *angle1array, double *angle2array, short *colarray, char *q)**

Same as *dv_p_arcs*, but each arc may have a different colour, specified by **colarray**. It occupies 4+50*npoints bytes.

int dv_p_arcfill(char *objectname, double x, double y, double width, double height, double angle1, double angle2, char *q)

Same as *dv_p_arc*, but here the arc is filled. It occupies 50 bytes.

int dv_p_arcsfill(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, double *angle1array, double *angle2array, char *q)

Same as *dv_p_arcs*, but here the arcs are filled. It occupies 4+48*npoints bytes.

int dv_p_arcscolfill(char *objectname, short npoints, double *xarray, double *yarray, double *widtharray, double *heightarray, double *angle1array, double *angle2array, short *colarray, char *q)

Same as *dv_p_arcscol*, but here the arcs are filled. It occupies 4+50*npoints bytes.

int dv_p_text(char *objectname, double x, double y, char *str,char *q)

Adds a picture element of type *text*. **x** and **y** define the left coordinates of the string **str**. It occupies 20 + strlen(str) + 1 bytes, plus one additional byte if strlen(str) is an even number.

int dv_p_textimage(char *objectname, double x, double y, char *str,char *q)

Same as *dv_p_text*, but also the background of the area occupied by the string will be affected (using the background colour). It occupies 20 + strlen(str) + 1 bytes, plus one additional byte if strlen(str) is an even number.

int dv_p_vertictext(char *objectname, double x, double y, char *str,char *q)

Same as *dv_p_text*, but the text is oriented vertically. The distance between each character is controllable via *dv_p_charsize* (default = 11 pixels).

int dv_p_vertictextimage(char *objectname, double x, double y, char *str,char *q)

Same as *dv_p_textimage*, but the text is oriented vertically. The distance between each character is controllable via *dv_p_charsize* (default = 11 pixels).

int dv_p_charsize(char *objectname, int charsize, char *q)

Defines the distance in pixels between characters for vertically oriented strings. The default value is 11 pixels. It occupies 4 bytes.

10.5.4 Functions to retrieve information about the picture object

int dv_p_getfirstcode(char *objectname, char *q)

Returns the **code** corresponding to the first element of a picture, and sets some internal counters to be used by *dv_p_getnextcode*. It returns -1 if the object does not exist or if it is empty.

int dv_p_getnextcode(char *objectname, char *q)

Returns the **code** of the next element of a picture. It returns -2 when the end of the picture is reached, or -1 in case of trouble. It can be called as many times as needed after a call to *dv_p_getfirstcode*.

int dv_p_getncodes(char *objectname, char *q)

Returns the **number of picture elements** defined in the object. Useful in combination with the next one.

int dv_p_getcodes(char *objectname, int *codearray, int *posarray, int maxelem, char *q)

It fills the user provided arrays **codearray** and **posarray** with *codes* and *positions* of the different picture elements contained by the object. **maxelem** specifies the maximum numbers of array elements to be filled.

int dv_p_getshort(char *objectname, int pos, short *svalue, char *q)

It puts in **svalue** the *short value* contained in the object memory at position **pos**. The position is expressed in bytes and has to be an even number. It returns 0 if everything is OK, -1 otherwise.

int dv_p_getdouble(char *objectname, int pos, double *dvalue, char *q)

It puts in **dvalue** the *double value* contained in the object memory at position **pos**. The position is expressed in bytes and has to be an even number. It returns 0 if everything is OK, -1 otherwise.

int dv_p_getstring(char *objectname, int pos, int maxchar, char *text, char *q)

It copies into **text** the *character string* (max. **maxchar** characters) contained in the object memory at position **pos**. The position is expressed in bytes and has to be an even number. It returns 0 if everything is OK, -1 otherwise.

10.5.5 Miscellaneous.

int dv_p_debug(char *objectname, int level, char *q)

If **level** > 0 switches on the printout of debug statements. If **level** == 0 it switches them off. It occupies 4 bytes.

int dv_p_analyse(char *objectname, char *q)

Produces a printout of the contents of the picture object. It does not add any byte to the picture object.

10.5.6 A function to build a picture reading from a text file

The instructions to build a picture are very easily written down in text format. Therefore we have added a function capable of interpreting the contents of a text file and building a picture out of it.

int dv_p_readfromfile(char *objectname, char *filename, char *q)

This function will read from the file **filename** the instructions defining a picture, and will invoke the corresponding functions described in the above sections. The file will contain three kind of lines :

- **remark** lines, starting by the ***** character, and ignored by the program .
- **instruction** lines, starting with a **keyword** and containing the **parameters** required for that keyword
- **data** lines, containing the data for multipoints elements.

Let us list the different keywords and parameters

GRAPH_LIMITS <graphname> <xmin> <ymin> <xmax> <ymax>
GRAPH_NAMES <graphname> <title> , <xlabel> , <ylabel>
VIEW_NAMES <viewname> <title> , <menuentry>
DEBUG <debug_level>
SETCNT <position>
CLEANCNT
COLOR <colorname>
BACKCOLOR <colorname>
LINESTYLE <linestylename>

LINEWIDTH <width>
MARKER <markername>
SIZETYPE <sizetypename>
FONTSIZE <fontsizename>
POINT <x> <y>
POINTS <npoints> , followed by npoints data lines <x> <y>
POINTSCOL <npoints> , followed by npoints data lines <x> <y> <colorname>
LINE <x1> <y1> <x2> <y2>
ARROW <x1> <y1> <x2> <y2>
LINES <npoints> , followed by npoints data lines <x> <y>
LINESCOL <npoints> , followed by npoints data lines <x> <y> <colorname>
SEGMENTS <npoints> , followed by npoints data lines <x1> <y1> <x2> <y2>
SEGMENTSCOL <npoints> , followed by npoints data lines <x1> <y1> <x2> <y2> <colorname>
DIAGONAL <x> <y> <width> <height>
DIAGONALS <npoints>, followed by npoints data lines <x> <y> <width> <height>
DIAGONALSCOL<npoints>, followed by npoints data lines <x> <y> <width> <height> <colorname>
RECTANGLE <x> <y> <width> <height>
RECTANGLES <npoints>, followed by npoints data lines <x> <y> <width> <height>
RECTANGLESCOL<npoints>, followed by npoints data lines <x> <y> <width> <height> <colorname>
RECTANGLEFILL <x> <y> <width> <height>
RECTANGLESFILL <npoints>, followed by npoints data lines <x> <y> <width> <height>
RECTANGLESCOLFILL <npoints>, followed by npoints data lines <x> <y> <width> <height> <colorname>
POLYGONFILL <npoints> , followed by npoints data lines <x> <y>
CIRCLE <x> <y> <ray>
CIRCLES <npoints>, followed by npoints data lines <x> <y> <ray>
CIRCLESCOL <npoints>, followed by npoints data lines <x> <y> <ray> <colorname>
CIRCLEFILL <x> <y> <ray>
CIRCLESFILL <npoints>, followed by npoints data lines <x> <y> <ray>
CIRCLESCOLFILL <npoints>, followed by npoints data lines <x> <y> <ray> <colorname>
ARC <x> <y> <width> <height> <angle1> <angle2>
ARCS <npoints>, followed by npoints data lines <x> <y> <width> <height> <angle1> <angle2>
ARCSCOL <npoints>, followed by npoints data lines <x> <y> <width> <height> <angle1> <angle2> <colorname>
ARCFILL <x> <y> <width> <height> <angle1> <angle2>
ARCSFILL <npoints>, followed by npoints data lines <x> <y> <width> <height> <angle1> <angle2>
ARCSCOLFILL <npoints>, followed by npoints data lines <x> <y> <width> <height> <angle1> <angle2> <colorname>
TEXT <x> <y> <freetext>
TEXTIMAGE <x> <y> <freetext>
VERTICTEXT <x> <y> <freetext>
VERTICTEXTIMAGE <x> <y> <freetext>
CHARSIZE <character size>

The following simple program will read a file, whose name is passed via command line argument, and try to build a picture interpreting its contents.

```
#include      <sys/types.h>
#include      <stdio.h>
#include      <sps.shm.h>
#include      "../dv_picture.h"
#include      "../dataviewer.h"
#include<math.h>
#define PICTURE "@/user/morpurgo/XSTUFF/DATAVIEWER_V6.4/test_picture/mops"
static char *q;
static char picturefile[80];
```

```
main(argc,argv)
int argc;
char **argv;
{
int i;
int status;
int dv_status;
int *dataptr;

strcpy(picturefile,argv[1]);
/*========MOPS CREATION AND INITIALIZATION======*/
c_sdkill(PICTURE);
q = c_sdalloc(PICTURE,100000L,SD_SHMALLOC);
c_sdini("PICTURE TEST",30L,q);
c_sdzero(10000,1,"picture","char",q);
c_sdquit(q);
q=c_sdacc(PICTURE,SD_RTC);
/*========SETTING UP DATAVIEWER VIEWS, GRAPHS, PLOTS ======*/
dv_init(5, 5, 5, q);
dv_status = dv_vwcreate("view1"," ", "", 1,1,q);
dv_status = dv_grcreate("graph1", "graph1","index","y",q);
dv_grattach("graph1", "view1",0,0,0,0,q);
dv_status = dv_plcreate1("plot1", DV_PICTURE, "picture", q);
dv_status = dv_plattach("plot1","graph1",q);
/*========   PICTURE CREATION=======*/
dv_p_readfromfile("picture",picturefile,q);
dv_p_analyse("picture",q);
c_sdquit(q);
}
```

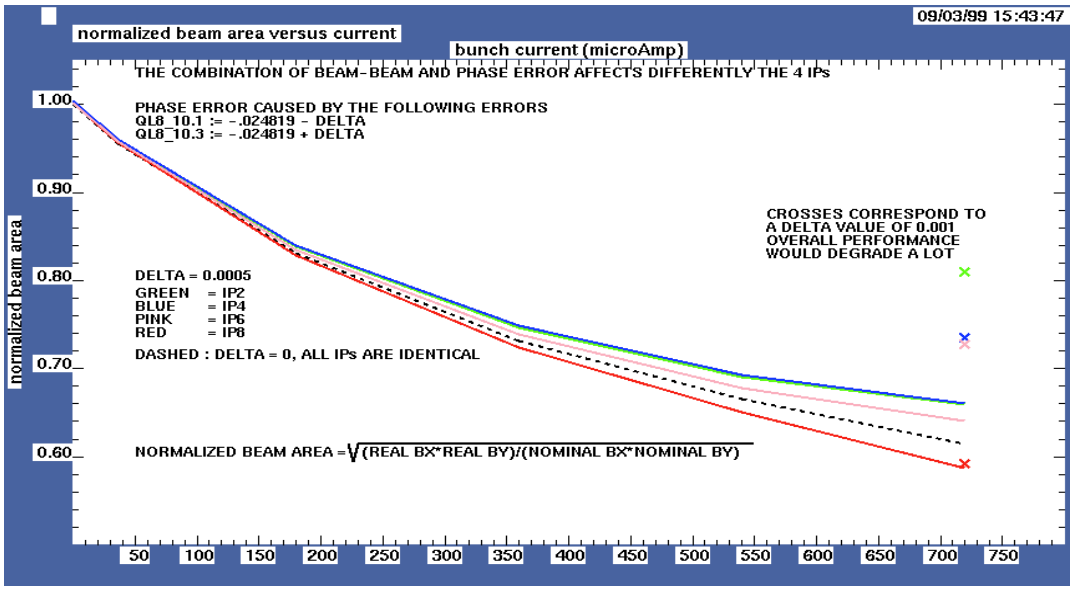

Figure 10.3. A picture generated by the Xdataviewer using a text file as input

And here we list the file used to generate Fig. 10.3:

```
GRAPH_LIMITS graph1 0 0.5 800.0 1.05
GRAPH_NAMES graph1 normalized beam area versus current , bunch current (microAmp), normalized beam area
MARKER CROSS
COLOR VISIBLE
LINESTYLE LINEONOFF
LINEWIDTH 2
LINES 6
```

```
  0.0   1.0
 36.0   0.956
180.0   0.831
360.0   0.732
540.0   0.665
720.0   0.615
LINESTYLE LINESOLID
COLOR RED
LINES 6
  0.0   1.001
 36.0   0.957
180.0   0.828
360.0   0.724
540.0   0.650
720.0   0.587
POINT 720.0 0.592
COLOR GREEN
LINES 6
  0.0   1.002
 36.0   0.959
180.0   0.839
360.0   0.747
540.0   0.691
720.0   0.659
POINT 720.0 0.810
COLOR BLUE
LINES 6
  0.0   1.004
 36.0   0.961
180.0   0.840
360.0   0.749
540.0   0.693
720.0   0.661
POINT 720.0 0.735
COLOR PINK
LINES 6
  0.0   1.002
 36.0   0.958
180.0   0.835
360.0   0.739
540.0   0.678
720.0   0.640
POINT 720.0 0.728
COLOR VISIBLE
FONTSIZE SMALLFONT
TEXT 50 .80 DELTA = 0.0005
TEXT 50 .78  GREEN
TEXT 50 .765 BLUE
TEXT 50 .75  PINK
TEXT 50 .735 RED
TEXT 110 .78  = IP2
TEXT 110 .765 = IP4
TEXT 110 .75  = IP6
TEXT 110 .735 = IP8
TEXT 50 .71  DASHED : DELTA = 0, ALL IPs ARE IDENTICAL
TEXT 560 .87   CROSSES CORRESPOND TO
TEXT 560 .855  A DELTA VALUE OF 0.001
TEXT 560 .84   OVERALL PERFORMANCE
TEXT 560 .825  WOULD DEGRADE A LOT
TEXT 50 0.99  PHASE ERROR CAUSED BY THE FOLLOWING ERRORS
TEXT 50 0.975 QL8_10.1 := -.024819 - DELTA
TEXT 50 0.96  QL8_10.3 := -.024819 + DELTA
TEXT 50 0.6 NORMALIZED BEAM AREA =   (REAL BX*REAL BY)/(NOMINAL BX*NOMINAL BY)
LINES 4
222 0.615
226 0.595
230 0.615
550 0.615
TEXT 50 1.03 THE COMBINATION OF BEAM-BEAM AND PHASE ERROR AFFECTS DIFFERENTLY THE 4 IPs
```

10.6 Appendix : where is it ?

in directory /user/bim/DATAVIEWER_V6.4 the programmer will find the following files (in addition to the standard Xdataviewer executable and libraries :

dv_picture.o : a library containing the functions described in this chapter.
dv_picture.h : a header file containing the definitions used in this chapter.

10.7 How to modify a picture element.

In this example, we show the correct way for modifying a picture element. We use the fact that the functions adding picture elements return the position of the element itself inside the Mops object. In our example we will change a COLOR Instruction from greeen to red.

```
.............
static int colorpos;
.............
main()
{
.............
build_picture();
.............
for(;;) {
  alarm = get_alarm();
   if (alarm) make_it(DV_RED);
   else make_it(DV_GREEN);
.............
   dv_kick();
}

int build_picture()
{
.............
colorpos = dv_p_color(“picture”,DV_GREEN,q); /* INITIALLY GREEN */
dv_p_point(“picture”,x,y,q);
dv_p_rectangle(“picture”,x,y,width,height,q);
.............
}

int make_it(
int color
)
{
int savecnt;

savecnt = dv_p_querycnt(“picture”,q);
dv_p_setcnt(“picture”, (short)colorpos, q);
dv_p_color(“picture”, color, q); /* set the new color */
dv_p_setcnt(“picture”,(short)savecnt, q);  /* restore the picture counter */
.................
}
```

11

The Help Facility for the Xdataviewer

11.1 The Help Facility

By clicking on the HELP button in the centre of the Dataviewer interface, a PullDown menu with several options will appear. By selecting one of these options, the User will be able to receive information about how to use the Xdataviewer, about the different Views, Graphs and Plots defined in this instance of the Dataviewer, and about the possible interactions between the Xdataviewer and the Application Program. The information concerning the Xdataviewer are kept in a file which should come together with the Xdataviewer executable itself. The information concerning the Application program must be provided by the Application Programmer. They will be contained in a file whose name will be communicated to the Xdataviewer via the routine

dv_set_help(filename,mopspointer)

described in chapter " The Xdataviewer Callable Interface". Inside this file, together with normal text lines, the application developer will insert "keyword lines", namely

- #VIEW <viewname>
- #GRAPH <graphname>
- #PLOT <plotname>
- #SELECT

which will be used by the Xdataviewer to search for the Help Information to be displayed depending on the Help Request. The following Requests are possible ;

- **General** : general information about the Xdataviewer
- **Message** : help on messages which can appear in the message field
- **Application** : the entire contents of the Application Help File
- **Views** : the contents of all the View sections in the Application Help File
- **CurrentView** : From the Application Help File : info about the currently selected View , its Graphs and its Plots.
- **Subview** : help on Subview selection
- **External** : help on the facilities contained in the External Pulldown Menu
- **Editor** : help on the editing facilities
- **Select** : From the Application Help File : information about the possible interactions between the Xdataviewer and the Application Program
- **Style** : help on the first four Option Menus in the third line of the interface
- **Zoom** : help on the Zoom facility
- **Cursor** : help on the information displayed on the cursor line.
- **Options** : help on the command line arguments acknowledged by the Xdataviewer program.

11.1.1 Example of Help File

General Info about the Application

.....

#VIEW myview1

Info for View "myview1"

#VIEW my view

Info for View "my view"

.....
#GRAPH graph1
Info for Graph “graph1”
.....
#PLOT dummyplot
Info for Plot “dummyplot”
.....
#SELECT
Info for the options under SELECT
.....

12

References and Acknowledgements

1. The Dataviewer Programmer's Guide. Ann Sweeney. CERN/SL/CO/Note/90-13

2. New features for the HP-UX Dataviewer. Giulio Morpurgo. Internal note.

3. M.O.P.S. User Guide for "C" programs. Werner Herr. CERN-SPS/88-43 (AMS), Revised January 1993

4. The CERN/SL Xdataviewer: An Interactive Graphical Tool for Data Visualization and Editing. Giulio Morpurgo. Proceedings of the ICAP98 Conference, Monterey (CA), September 1998

5. Xcreator : a C code generator for AppliXation Programs. Giulio Morpurgo . Internal note.

6. Xcreator Auxiliary User Guide. Giulio Morpurgo. Internal note.

7. A picture drawing facility for the Xdataviewer. Giulio Morpurgo. Internal note.

It is my pleasure to thank the many people who have contributed to the success of this project. By looking back into history, one finds that the data viewer concept was originally developed by C.Saltmarsh, I.Gjerpe and R.Lauckner. I.Gjerpe actually implemented the first version, which was later extended by D.Brignon, L.Normann, J.Miles, W.Herr, J.Ulander, I.Wilkie and A.Ogle. But the biggest acknowledgement goes to Ann Sweeney, who did an excellent job, synthesizing all these earlier developments into a well documented and structured piece of code. A second thanks goes to Alan.J. Burns who, as my supervisor, strongly encouraged and supported my determination to work on this project. Many colleagues of mine have influenced and provided feedback to the development of the Xdataviewer, and have always been very active in testing and using the latest releases. I would like to thank in particular G. Crockford, J.J.Gras, W. Herr, M. Jonker, J.C.Oliveira, V. Paris and, not to forget anyone, the all SPS and LEP Operators, who have to survive with this product in their daily duty.

Appendix 1

Where to find the Xdataviewer

At CERN, in directory **/user/bim/DATAVIEWER** on the HP-UX machine "**bomlep**", one can find the following files :

- **Xdataviewer** : the executable program
- **dataviewer** : a link to Xdataviewer, maintained for historical reasons
- **dv_user.o** : the Callable Interface for Application Programs
- **dv_embex.a** : the library needed by Applications using the Embedded Dataviewer
- **dataviewer.h** : header file containing definitions used by the Callable Interface
- **dv_picture.o** : contains the routines for the picture drawing facility
- **dv_picture.h** : contains definition used inside the routines in dv_picture.o

The Embedded Dataviewer User also needs a few files from the Xcreator product. The following files will be found in directory **/user/bim/XCREATOR/lib** :

- **Xcreator_lib.o**
- **Xcreator_enter_leave.o**

Finally, the MOPS library is installed as library **libshm.a** under **/usr/opt/MOPS/lib** , and its header file is **sps.shm.h** in directory **/usr/opt/MOPS/include**

Examples and makefiles can be found in directories **tests**, **test_embed**, **testdoc**, **test_picture** under **/user/morpurgo/XSTUFF/DATAVIEWER_V6.4** (on the HP-UX machine called "**hpslz14**").

Support requests , questions and other remarks should be sent to the following email address :

giulio.morpurgo@cern.ch

Appendix 2

Colours and Markers used by the Xdataviewer

All the constant definitions useful for the Application Programmer are contained in the header files “dataviewer.h” and “dv_picture.h” (for the picture drawing facility).

2.1 Colours.

The following colour definitions can be used in the Xdataviewer routines and in filling colour objects ;

0 = DV_BLACK	8 = DV_NAVY	16 = DV_WHEAT	24 = DV_INDIANRED
1 = DV_WHITE	9 = DV_ORANGE	17 = DV_PURPLE	25 = DV_KHAKI
2 = DV_RED	10 = DV_PINK	18 = DV_AQUAMARINE	26 = DV_PLUM
3 = DV_GREEN	11 = DV_BROWN	19 = DV_SALMON	27 = DV_SEAGREEN
4 = DV_BLUE	12 = DV_GREY	20 = DV_ORCHID	28 = DV_OLIVE
5 = DV_CYAN	13 = DV_LIMEGREEN	21 = DV_GOLD	29 = DV_TOMATO
6 = DV_MAGENTA	14 = DV_TURQUOISE	22 = DV_STEELBLUE	30 = DV_CORAL
7 = DV_YELLOW	15 = DV_VIOLET	23 = DV_LIGHTBLUE	31 = DV_SKY

We remind that, by default, DV_WHITE is always visible (white on a black background and black on a white background) and DV_BLACK always invisible (the opposite as DV_WHITE).
To make DV_BLACK points visible and DV_WHITE points invisible one could use the command line argument “-useblack”.
More conveniently, two “metacolors” can also be used : 98 = DV_VISIBLE and 99 = DV_INVISIBLE.
It is possible to produce a grey scale display using codes from 100 = DV_GREY_START to 131 = DV_GREY_STOP .

Finally, notice that the availability of all the above mentioned colours depends also on the resources of your X Terminal, for which the Xdataviewer has to compete with other running Applications (like Netscape, FrameMaker, FaceMaker, etc.).

2.2 Markers

The following marker types are defined :

0 = DV_NONE (invisible)	8 = DV_UP_ARROW	16 = DV_RIGHT_ARROW
1 = DV_POINT	9 = DV_GIULIO_DOT	17 = DV_HOR_DOT
2 = DV_DOT	10 = DV_HUGE_DOT	18 = DV_VER_DOT
3 = DV_PLUS	11 = DV_MIDDLE_DOT	19 = DV_BEND
4 = DV_BOX	12 = DV_HOR_BOX	
5 = DV_CROSS	13 = DV_VER_BOX	20 = DV_USER_DOT
6 = DV_DEREK_DOT	14 = DV_ROMBOID	21 = DV_USER_BOX
7 = DV_DOWN_ARROW	15 = DV_LEFT_ARROW	22 = DV_USER_CIRCLE
		23 = DV_USER_ARC

The User can define the size in pixels for the marker types DV_USER_xxx . Figure A2.1 shows the different marker types (a size = 3 is used for the DV_USER_xxx markers).

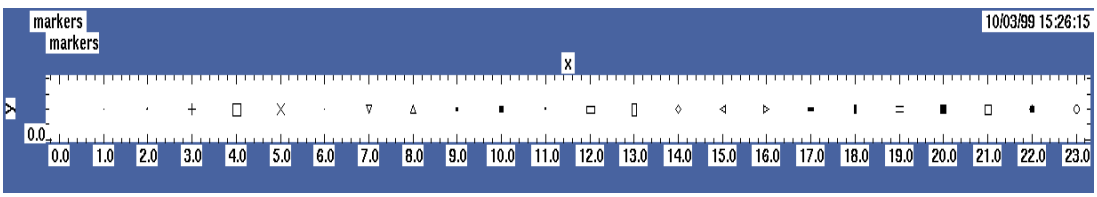

Figure A2.1. The different marker types (DV_NONE is invisible)

Appendix 3

Xdataviewer Error Codes

The following error codes can be returned by the routines in the Callable Interface

0 = DV_OK
1 = DV_NO_ROOM
2 = DV_VIEW_EXISTS
3 = DV_GRAPH_EXISTS
4 = DV_PLOT_EXISTS
5 = DV_NO_SUCH_VIEW
6 = DV_NO_SUCH_GRAPH
7 = DV_NO_SUCH_PLOT
8 = DV_NO_PID
9 = DV_NO_SUCH_TYPE
10 = DV_NO_SELECTION
11 = DV_INVALID_OPTION
12 = DV_NO_COMM
13 = DV_ILLEGAL
14 = DV_OUT_OF_RANGE
101 = DV_NO_SHARDAT
102 = DV_CANNOT_CREATE_FILE
103 = DV_SDINI_ERROR
104 = DV_SHM_ALLOCATION_ERROR
109 = DV_SDWRIT_ERROR
110 = DV_SDQUIT_ERROR
111 = DV_CALLOC_ERROR
112 = DV_SDKILL_ERROR
113 = DV_INVALID_MODE
114 = DV_SDACC_ERROR
115 = DV_NO_SUCH_OBJECT
116 = DV_SDCUT_ERROR
117 = DV_SDPTR_ERROR
118 = DV_CANNOT_OPEN_FILE

Appendix 4

Minimal MOPS C-Library Reference

While a complete documentation for the MOPS library has to be found in reference 3, it is worth here listing the most used routines, together with their calling sequences.

CREATING A MOPS

char *__c_sdalloc__ (char *name, long size, long flag)

- **name** is the name of the dummy file to identify the Unix shared memory segment . The file will be looked for in $HOME/shardat unless the first character of name is @. In this case, this character will be skipped, and the remaining part of name will be used as a complete path to the dummy file. The dummy file must already exist.
- **size** is the size of the memory buffer to be reserved for the MOPS
- **flag** is one of SD_SHMALLOC (real Unix shared memory) or SD_MEMALLOC (memory buffer dynamically allocated in user process space)

The routine creates the MOPS, which should not exist before, and returns a pointer to it, to be used in most of the other calls. It returns a (character pointer casted) value of -1 to -5 in case of error.

```
ex.     system("touch  /usr/tmp/mymops");
        q = c_sdalloc("@/usr/tmp/mymops",300000,SD_SHMALLOC)
```

ACCESSING A MOPS

- char *__c_sdacc__ (char *name, long flag)
- **name** as above
- **flag** is one of SD_WRITE, SD_READ, SD_WRITE_NOWAIT, SD_RTC .

The routine returns a pointer to the MOPS, or a (character pointer casted) value of -1 to -3 in case of error. In case of success, the access rights will depend on the chosen flag. If SD_READ is used, the process will not be able to modify the MOPS. SD_WRITE uses a semaphore to protect the MOPS, while SD_RTC accesses the MOPS without checking for other users.

```
ex.     q = c_sdacc("@/usr/tmp/mymops",SD_RTC)
```

INITIALIZING A MOPS

long __c_sdini__ (char * mopsname, long nobj, char *q)

- **mopsname** is a name to be stored internally to the MOPS. it does not need to be the same as the name in c_sdalloc or c_sdacc .
- **nobj** is the number of user objects slots which the Application wants to define in the MOPS. If the Xdataviewer is used, 5 of these slots will be used by it.
- **q** is the pointer to the MOPS, returned by one of *c_sdalloc*, *c_sdacc*. One can also create a MOPS inside a normal memory buffer, and in this case a pointer to the buffer can be used instead. The same pointer will be used in most of the Xdataviewer calls.

This routine structures a memory buffer as a MOPS, reserving the space for all the directory entries corresponding to the different objects to be defined. These entries will be later filled by *c_sdbook* or *c_sdzero*. 5 entries will be occupied by *dv_init*, so the Programmer should declare at least 5 entries more than the data objects created by his application.

```
ex.     c_sdini("this is my mops", 40, q)
```

The routine returns *nobj* if successful, -1 otherwise.

CREATING A MOPS OBJECT

long __c_sdbook__ (long nelems, long elemsz, char *objname, char *objtype, char *q)
long __c_sdzero__ (long nelems, long elemsz, char *objname, char *objtype, char *q)

- **nelems** is the number of elements in the object

- **elemsz** is the size of one elements (in bytes)
- **objname** is the name of the object (later used to access and refer to it)
- **type** is the element type (e.g. “long”, “int”, “float”...)
- **q** is the pointer to the MOPS (same as in *c_sdini*)

These routines fill one of the MOPS directory entries with the features of the object to be created, and reserve the necessary memory space in the data part of the MOPS. *c_sdzero* also sets all this space to zero.
ex. c_sdbook(100,4, “my_integer”,”int”,q)

The routines return the index number of the object allocated, or a negative value in case of error.

ACCESSING A MOPS OBJECT
char * **c_sdptr** (char *objname, char *q)

- **objname** is the name of the object to be accessed
- **q** is the pointer to the MOPS (same as in *c_sdini*)

Returns a pointer to the beginning of the data space for the object *objname*. This pointer (casted in the proper way) can be treated as a normal array pointer.
ex. int *my_integer;
.....
my_integer = (int *)c_sdptr(“my_integer”,q)

It returns a pointer to a value of -1 or -2 in case of error.

COPYING AN ARRAY INTO A MOPS OBJECT
long **c_sdwrit** (char *objname, long nelems, char *data, char *q)

- **objname** is the object whose elements have to be written
- **nelems** is the number of elements to be written
- **data** is a pointer to the user array containing the data to be written in the MOPS object
- **q** is the pointer to the MOPS (same as in *c_sdini*)

Used to copy *nelems* of a user supplied array into a MOPS object. It returns the number of bytes written, or a negative value in case of error.
ex. int array[100]
.....
for (i=0;i<100;i++) array[i] = i;
c_sdwrit(“my_integer”,100,(char *)array,q);

SAVING THE MOPS INTO A BINARY FILE
long **c_sdfil** (char *filename, char *q)

- **filename** is the name of a file into which the MOPS has to be saved.
- **q** is the usual MOPS pointer

The routine copies the complete MOPS memory into a binary file. It returns the number of bytes written, or a negative number in case of error.

QUERYING ABOUT A MOPS EXISTENCE
long **c_sdq** (char *name)

- **name** is the name of the dummy file associated with the MOPS (see *c_sdalloc*)

The routine returns 1 if such a MOPS exists, 0 if the dummy file exists but the MOPS does not, -1 if the dummy file does not exist.

TERMINATING ACCESS TO A MOPS

long **c_sdquit** (char *q)

- **q** is the usual MOPS pointer

Terminates the access to the MOPS. After this call, *q* is no longer valid, and the MOPS must be reaccessed before being used again. Also the individual objects must be reaccessed using *c_sdptr* .

REMOVING A MOPS FROM THE COMPUTER

long **c_sdkill** (char *name)

- **name** is the name of the dummy file associated with the MOPS (see *c_sdalloc*)

This routine tries to remove the specified MOPS from the computer memory. In case of success, after the call the shared memory segment and its associated semaphore do not exist anymore.

Two useful commands
direc <mopsname> and **rem** <mopsname>
to be found in **/usr/opt/MOPS/bin**
respectively enable the User to list the contents of a MOPS directory and to remove the MOPS from the computer.

Appendix 5

Xdataviewer Callable Interface Quick Reference

This appendix contains a list of the routines described in detail in chapter 3 . The most important routines appear in bold characters.

XDATAVIEWER ROUTINES

INITIALIZATION OPERATIONS

int **dv_init** (int no_views, int no_graphs, int no_plots, char *q)
int dv_space (int no_views, int no_graphs, int no_plots, int *space_needed)
int dv_delentry (int entrytype, char *entryname, char *q)
int dv_set_help (char *filename, char *q)
int dv_set_view (char *viewname, char *q)
int dv_set_view_and_graph (char *viewname, char *graphname, char *q)

ROUTINES MODIFYING THE XDATAVIEWER BEHAVIOR

int **dv_set_update_mode** (int mode, char *q)
int dv_set_raising_mode (int mode, char *q)

ROUTINES RELATED TO SAVING THE CONTENTS OF A VIEW BACK INTO THE DATA SPACE

int dv_set_save_request (char *q)
int dv_status_set (int option, int value, char *q)
int dv_status_get (int option, char *q)

ROUTINES ACTING ON THE XDATAVIEWER GRAPHICAL INTERFACE

int dv_mode_set (int option, int value, char *q)
int dv__mode_set (int option, int value) (for Embedded Dataviewer)
int dv_set_external_menu_string (int option, char *optionstring, char *q)

ROUTINES FOR APPLICATION->XDATAVIEWER INTERACTION

int **dv_kick** (char *q) for standalone Xdataviewer
int **dv__kick**() for Embedded Dataviewer
int dv_kick_and_save (char *q) for standalone Xdataviewer
int dv__kick_and_save() for Embedded Dataviewer
int dv_get_dv_processid (int *dv_processid, *char *q)
int set_dv_app_processid (int app_processid, char *q)
int **dv_set_selection_menu** (int noptions, char options[][16])
int dv_set_selection_prompt (int noption, char *prompt, char *q)
int **dv_get_selection** (int *option, char *viewname, char *graphname, char *plotname, char *xdataname, char *ydataname, char *textname, int *index, double *xvalue, double *yvalue, char *textstring, char *q)
int dv_get_selection_cursor (double *x, double *y, char *q)

FORCING XDATAVIEWER TO REPLACE THE DATA AREA SPACE WITH A NEW ONE

int **dv_newdata_set** (char *newdata_name, char *q)

OPERATIONS WITH A MOPS CONTAINED IN BINARY FILES

int **dv_save_file** (char *filename, char *q)
char ***dv_attach_file** (char *filename)
char *dv_attach_filegrow (char *filename, int size, int mode)
int dv_save_filecomm (char *filename, char *q)
int **dv_quit_file** (char *q)
int **dv_kick_file** (char *q)
int **dv_set_enablewriting** (int mode, char *q)

DEFAULT SETTINGS FOR GRAPH AND PLOT CREATION

int dv_set_default (char *name, long value)

int dv_set_defaultlimit (char *name, double value)
int dv_set_defaultformat (char *name, char *value)
int dv_show_defaults()

VIEW CREATION
int **dv_vwcreate** (char *viewname, char *viewtitle, char *menuentry, int no_rows, int no_cols, char *q)
int dv_vwchnames (char *viewname, char *viewtitle, char *menuentry, char *q)
int dv_vwchtitle (char *viewname, char *viewtitle, char *q)
VIEW AVAILABILITY
int dv_vwhide (char *matchname, int mode, char *q)
int dv_vwshow (char *matchname, int mode, char *q)

GRAPH CREATION AND ATTACHEMENT
int **dv_grcreate** (char *graphname, char *title, char *xlabel, char *ylabel, char *q)
int **dv_grattach** (char *graphname, char *viewname, int rowpos, int colpos, int rowsize, int colsize, char *q)
int dv_grdetach (char *graphname, char *viewname, char *q)

SETTING AND MODIFYING GRAPH PROPERTIES
int dv_grchnames (char *graphname, char *title, char *xlabel, char *ylabel, char *q)
int dv_grlabels (char *graphname, int nlabels, char labels[][80], char *q)
int dv_grlimits (char *graphname, double xmin, double xmax, double ymin, double ymax, char *q)
int dv_grpercent (char *graphname, double xpercent, double ypercent, char *q)
int dv_grscales (char *graphname, int xscale, int yscale, char *q)
int dv_grsetgrid (char *graphname, int grid, char *q)
int dv_grsetzeroline (char *graphname, int zeroline, char *q)

LISTING A GRAPH IN TEXT FORMAT
int dv_grlist (char *graphname, int listopt, char *q)
int dv_grlistmerge (char *graphname, int listopt, char *q)
int dv_grliststyle (char *graphname, int liststyle, char *q)

CONTROLLING THE NUMBER OF POINTS TO BE DISPLAYED FOR A GRAPH
int dv_grnpoints (char *graphname, int npoints, char *q)

PLOT CREATION AND ATTACHEMENT
int **dv_plcreate1** (char *plotname, int plottype, char *ydata, char *q)
int **dv_plcreate2** (char *plotname, int plottype, char *xdata, char *ydata, char *q)
int **dv_plattach** (char *plotname, char * graphname, char *q)
int dv_pldetach (char *plotname, char * graphname, char *q)

SPECIFYING THE DATA CHARACTERISTICS FOR A PLOT
dv_pldtform (char *plotname, int dataform, char *q)

GIVING COLOR TO THE PLOTS
int dv_plhis (char *plotname, int histcol, char *q)
int dv_pllin (char *plotname, int linecol, char *q)
int dv_plmar (char *plotname, int markercol, int markertype, char *q)
int dv_plcol (char *plotname, char *colourname, char *q)
int dv_plcolouredline (char *plotname, int flag, char *q)

ASSIGNING LABELS TO PLOT POINTS
int dv_pltxt (char *plotname, char *text, char *q)

ASSIGNING ERROR BARS TO PLOT POINTS
int dv_plerrorbars (char *plotname, char *x_left, char *x_right, char *y_top, char *y_bottom,char *q)

LISTING A PLOT IN TEXT FORMAT

int dv_pllist (char *plotname, int listopt, char *q)
int dv_plcoltxt (char *plotname, char *colourname, char *q)
int dv_plprecision (char *plotname, int precision_x, int precision_y, char *q)
int dv_pllistform (char *plotname, char *format_x, char *format_y, char *q)

HOW TO MAKE A PLOT EDITABLE
int **dv_plreadwrite** (char *plotname, int readwrite, char *q)

CONTROLLING THE NUMBER OF POINTS TO BE DISPLAYED FOR A PLOT
int dv_plnpoints (char *plotname, int npoints, int first, char *q)
int dv_ObjectResize (char *objectname, int no_elements, char *q)

STORING AND RETRIEVING INDIVIDUAL DATA OBJECTS FROM FILES
int dv_file_object (char *objectname, char *filename, char *q)
int dv_file_array (char *buf, long nelems, long type, char *filename)
int dv_read_object (char *buf, long maxelem, char *filename)
int dv_seek_object (char *filename, long *nelems, long *elemsz, long *elecod)

QUERY ROUTINES
GENERAL CONFIGURATION QUERIES
int dv_qviews (int *no_views, char **viewnames[], char *q)
int dv_qgraphs (int *no_graphs, char **graphnames[], char *q)
int dv_qplots (int *no_plots, char **plotnames[], char *q)

VIEW SPECIFIC QUERIES
int dv_qvwexist (char *viewname, int *exists, char *q)
int dv_qvwcontents (char *viewname, int *no_graphs, char **graphnames[], int *rowpos[], int *colpos[], int *rowsize[], int *colsize[], char *q)
int dv_qvwcreate (char *viewname, char *title, char *menuentry, int *no_rows, int *no_cols, char *q)

GRAPH SPECIFIC QUERIES
int dv_qgrexist (char *graphname, int *exists, char *q)
int dv_qgrattachments (char *graphname, int *no_views, char **viewnames[], int *rowpos[], int *colpos[], int *rowsize[], int *colsize[], char *q)
int dv_qgrcontents (char *graphname, int *no_plots, char **plotnames[], char *q)
int dv_qgrcreate (char *graphname, char *title, char *xlabel, char *ylabel, char *q)
int dv_qglabels (char *graphname, int *nlabels, char **labels[], char *q)
int dv_qgrlimits (char *graphname, double *xmin, double *xmax, double *ymin, double *ymax, char *q)
int dv_qgrpercent (char *graphname, double *xpercent, double *ypercent, char *q)
int dv_qgrscales (char *graphname, int *xscale, int *yscale, char *q)
int dv_qgrlist (char *graphname, int *listopt, int *mergeflag, char *q)
int dv_qgrliststyle (char *graphname, int *liststyle, char *q)
int dv_qgrnpoints (char *graphname, int *npoints, char *q)
int dv_qgrsetgrid (char *graphname, int *gridstyle, char *q)
int dv_qgrsetzeroline(char *graphname, int *zeroline, char *q)

PLOT SPECIFIC QUERIES
int dv_qplexist (char *plotname, int *exists, char *q)
int dv_qplattachments (char *plotname, int *no_graphs, char **graphnames[], char *q)
int dv_qplcreate (char *plotname, int *plottype, char *xdata, char *ydata, char *q)
int dv_qpldtform (char *plotname, int *dataform, char *q)
int dv_qplreadwrite (char *plotname, int *readwrite, char *q)
int dv_qplhis (char *plotname, int *histcol, char *q)
int dv_qpllin (char *plotname, int *linecol, char *q)
int dv_qplmar (char *plotname, int *markercol, int *markertype, char *q)
int dv_qplcol (char *plotname, char *colours, char *q)
int dv_qplcolouredline (char *plotname, int *value, char *q)
int dv_qplcoltxt (char *plotname, char *colours, char *q)

int dv_qpltxt (char *plotname, char *text, char *q)
int dv_qplerrorbars (char *plotname, char *x_left, char *x_right, char *y_top, char *y_bottom, char *q)
int dv_qpllist (char *plotname, int *listopt, char *q)
int dv_qpllistform (char *plotname, char *format_x, char *format_y, char *q)
int dv_qplnpoints (char *plotname, int *npoints, int *first, char *q)

MISCELLANEOUS QUERIES
int dv_qobexist (char *obname, int *exists, int *type, char *q)
int dv_qversion (int *version , char *q)

EMBEDDED DATAVIEWER ROUTINES
ROUTINES TO SPECIFY DATA TO BE LOADED
int **dv__sharename** (char *mopsname)
int **dv__pointer** (char *ptr)
int **dv__filename** (char *filename)
int dv__sequencename (char *sequencename)
INITIALIZATION
int **dataviewer_initialize**()
int **dv_before_main**()
FREEING UNUSED SPACE
int dv__clearchangeptr ()
int dv__freeptr (char *ptr)
OPTION SPECIFICATION FOR THE EMBEDDED DATAVIEWER
int dv__initialview (char *viewname)
int dv__initialgraph (char *graphname)
int dv__argument (char * optionname, char *optionvalue)
COMMUNICATION BETWEEN THE APPLICATION AND THE EMBEDDED DATAVIEWER
int **dv__kick**() : discussed in section 3.2
int dv__kick_and_save() : discussed in section 3.2
int dv__mode_set (int option, int value) : discussed in section 3.1
int dv__gimme_window()

DEBUGGING
void dv_error (int error)
SAVING AND RESTORING XDATAVIEWER CONFIGURATION
int dv_confsave (filename, q)
int dv_confrestore (filename, q)

OBSOLETE ROUTINES, NOT TO BE USED
int dv_dsattach (char *data_area_name, int mode, char **q)
int dv_dserase (char *data_area_name)
int dv_dsrelease (char *q)
int dv_obsize (int size)
int dv_dsinit (char *data_area_name, int no_views, int no_graphs, int no_plots, int no_objects, char **q)
int dv_dsdisplay (char *viewname, char *q)

MOPS ROUTINES (FROM APPENDIX 4)
char *__c_sdalloc__ (char *name, long size, long flag)
char *__c_sdacc__ (char *name, long flag)
long __c_sdini__ (char * mopsname, long nobj, char *q)
long __c_sdbook__ (long nelems, long elemsz, char *objname, char *objtype, char *q)
long c_sdzero (long nelems, long elemsz, char *objname, char *objtype, char *q)
char * __c_sdptr__ (char *objname, char *q)
long c_sdwrit (char *objname, long nelems, char *data, char *q)
long c_sdfil (char *filename, char *q)
long c_sdq (char *name)
long __c_sdquit__ (char *q)
long __c_sdkill__ (char *name)

Appendix 6

Examples of Xdataviewer displays from real Applications.

Just to give a flavour of what is possible to achieve using the Xdataviewer, we report a few examples.

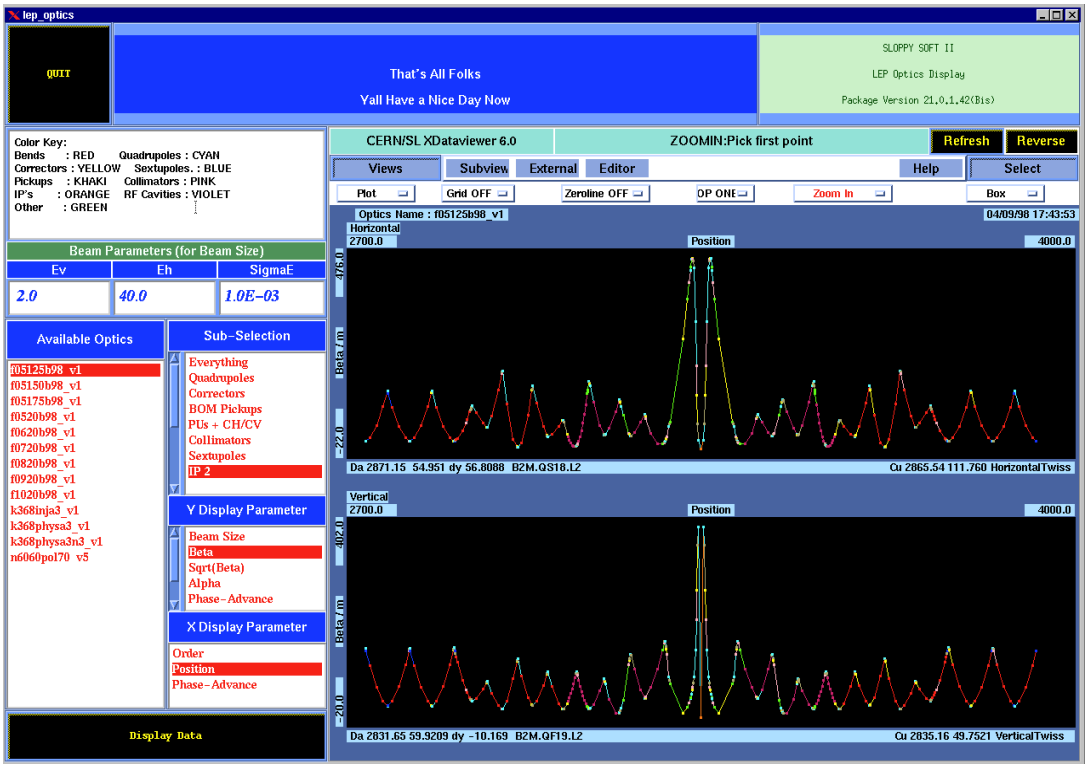

Figure A6.1. The Optics Display programs using the Embedded Dataviewer. Every element of LEP (dipoles, quadrupoles, rf, etc.) is represented using a different color .

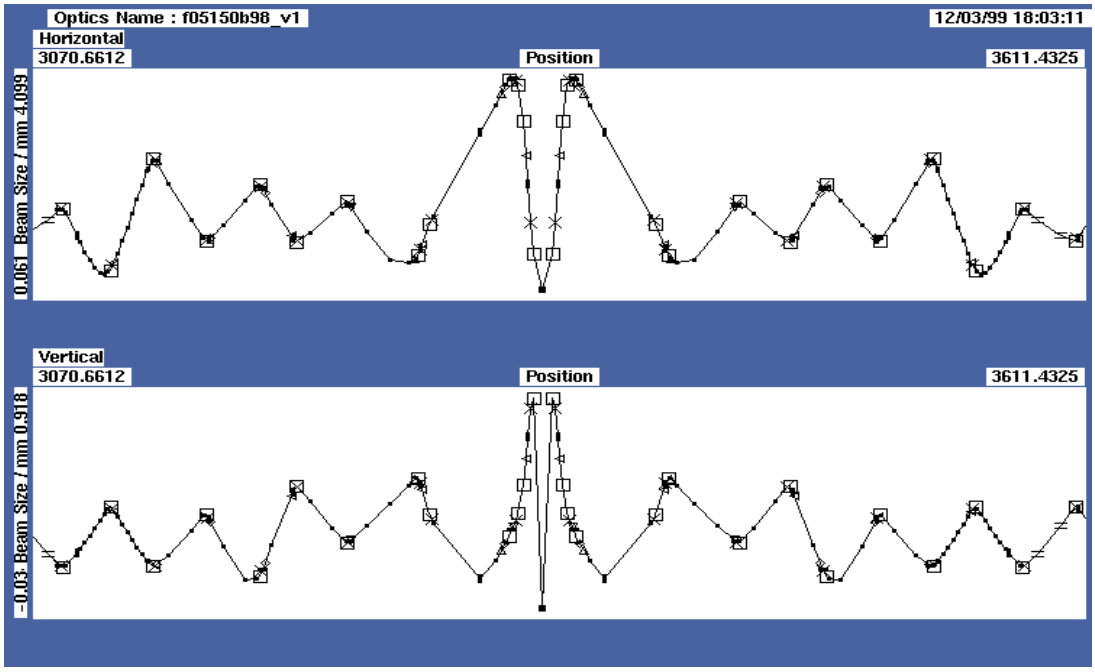

Figure A6.2 . The same data as above displayed as a MULTIMARKERLINE plot. This time every element is represented using a different Marker type.

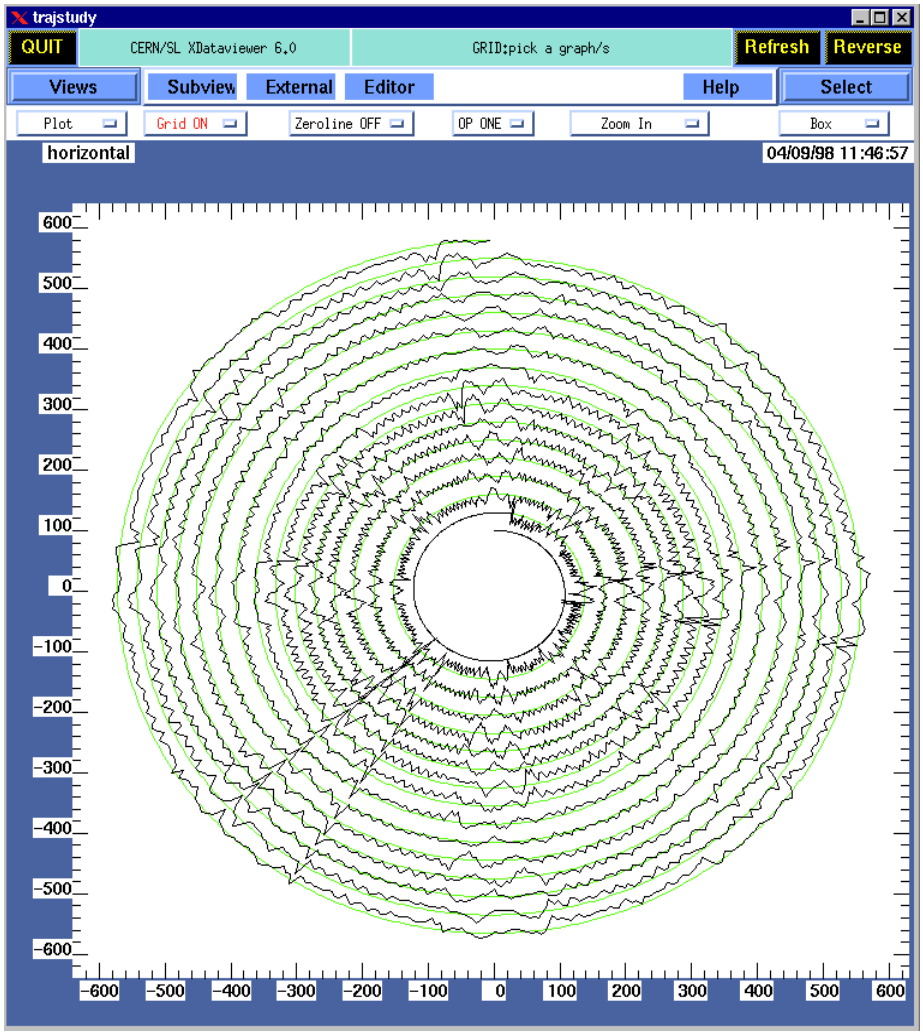

Figure A6.3. The first 15 turns of electrons in LEP, represented as a spiral (application Trajstudy). The graph contains two 2-dim plots : the spiral representing the center of the vacuum chamber, and the trajectory data, oscillating around it.

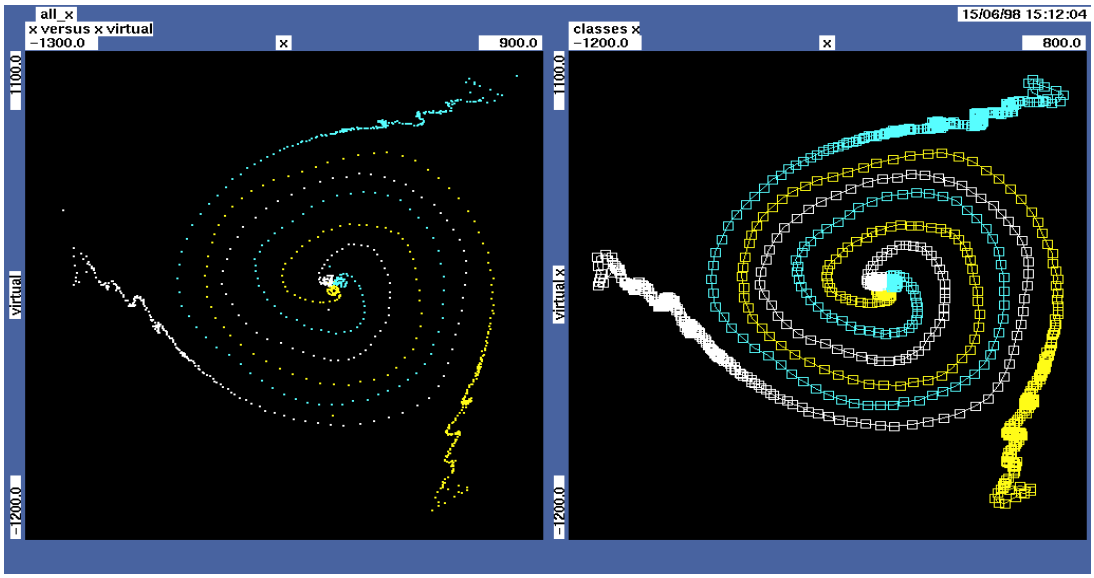

Figure A6.4 . Visualization of the phase space in LEP. The beam was kicked near the third order resonance.

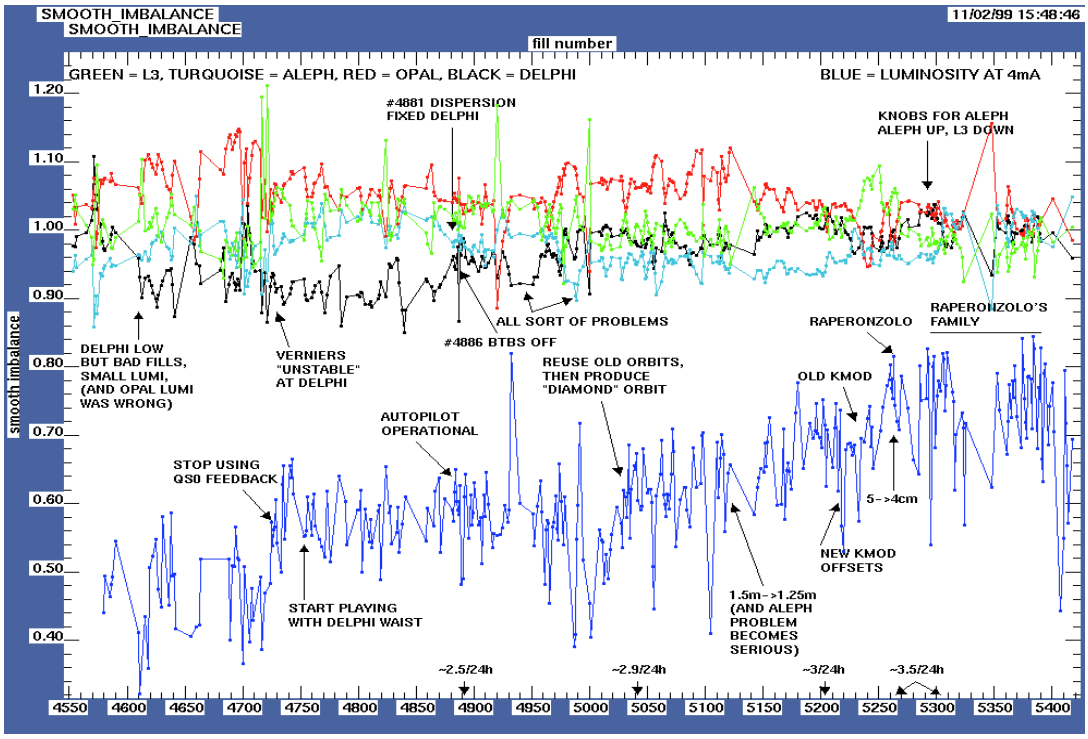

Figure A6.5. A graph containing 5 data plots of type DV_LINE, and a DV_PICTURE plot. The DV_PICTURE plot contains all the text and arrows definitions.

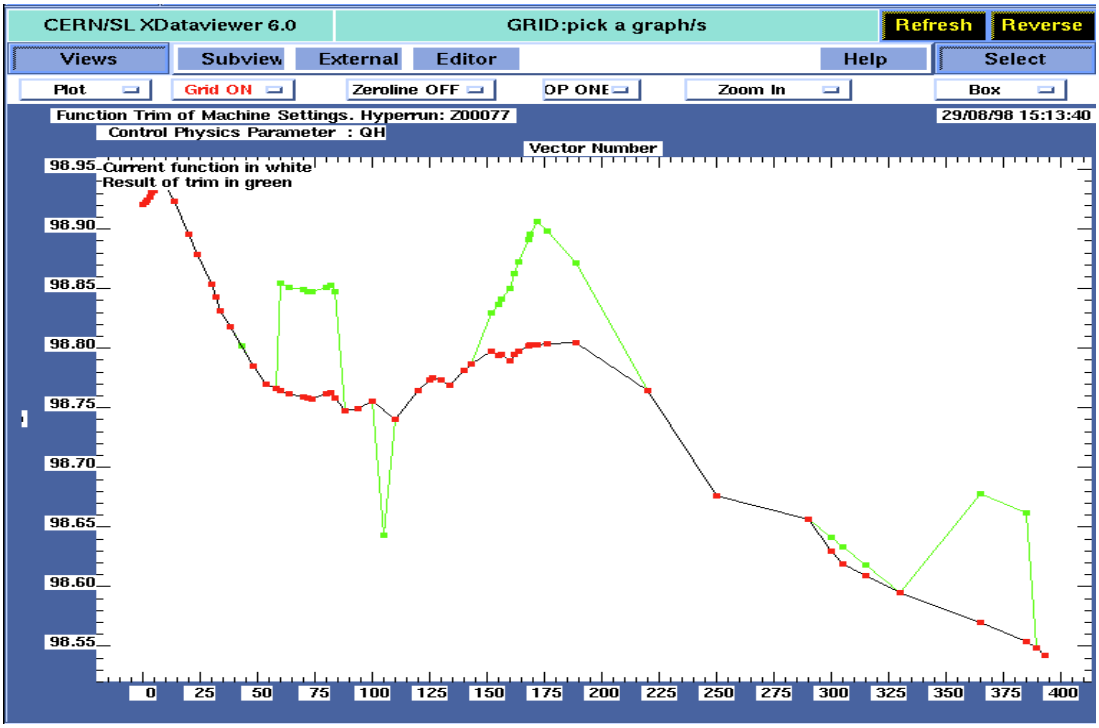

Figure A6.6. Editing operations on a plot (from the Function Editor application).